Titre: Title:	Méthodes sans factorisation pour l'optimisation non linéaire
Auteur: Author:	Sylvain Arreckx
Date:	2016
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Arreckx, S. (2016). Méthodes sans factorisation pour l'optimisation non linéaire [Thèse de doctorat, École Polytechnique de Montréal]. PolyPublie. https://publications.polymtl.ca/2213/

UNIVERSITÉ DE MONTRÉAL

MÉTHODES SANS FACTORISATION POUR L'OPTIMISATION NON LINÉAIRE

SYLVAIN ARRECKX
DÉPARTEMENT DE MATHÉMATIQUES ET DE GÉNIE INDUSTRIEL
ÉCOLE POLYTECHNIQUE DE MONTRÉAL

THÈSE PRÉSENTÉE EN VUE DE L'OBTENTION
DU DIPLÔME DE PHILOSOPHIÆ DOCTOR
(MATHÉMATIQUES DE L'INGÉNIEUR)
AOÛT 2016

UNIVERSITÉ DE MONTRÉAL

ÉCOLE POLYTECHNIQUE DE MONTRÉAL

Cette thèse intitulée :

MÉTHODES SANS FACTORISATION POUR L'OPTIMISATION NON LINÉAIRE

présentée par : ARRECKX Sylvain
en vue de l'obtention du diplôme de : Philosophiæ Doctor
a été dûment acceptée par le jury d'examen constitué de :

M. LODI Andrea, Doctorat ingénierie, président
M. ORBAN Dominique, Doctorat, membre et directeur de recherche
M. ANJOS Miguel F., Ph. D., membre
M. BIRGIN Ernesto G., Ph. D., membre externe

DÉDICACE

Aux gens qui ont cru en moi et qui m'ont soutenu

REMERCIEMENTS

Je tiens tout d'abord à remercier mon directeur de thèse, Dominique Orban, pour son aide, sa patience et ses conseils précieux tout au long du processus ayant mené à la réalisation de cette thèse.

Ce travail n'aurait pas été possible sans l'aide et la présence de nombreuses personnes qui ont toutes eu une importance non négligeable sur moi ou ma façon de travailler. Je remercie chacune d'entre elles chaleureusement.

Je remercie notamment le GERAD pour les possibilités de recherche qu'il offre et pour la disponibilité de son personnel et de ses membres. À l'École Polytechnique, je tiens particulièrement à remercier le département de Mathématiques et Génie Industriel, pour les possibilités d'enseignement qui m'ont été présentées ainsi que pour le soutien offert. Ces expériences enrichissantes d'enseignement m'ont beaucoup apporté personnellement et professionnellement.

Un grand merci aussi à Antoine DeBlois pour son encadrement durant mon stage MITACS au département d'aérodynamique avancée de Bombardier, ainsi qu'à tous les membres de cette équipe pour m'avoir permis de découvrir le monde de l'aérodynamique et les challenges relatifs à l'optimisation multidisciplinaire dans un environnement industriel.

Je souhaite aussi remercier les organismes qui m'ont soutenu financièrement pendant mon doctorat : le CRSNG, MITACS et Bombardier.

J'adresse mes remerciements à Andrea Lodi, Miguel Anjos, Ernesto Birgin et Eric Laurendeau pour avoir accepté de faire partie de mon jury de thèse.

Je souhaite remercier mes parents et mes frères et sœurs pour leur soutien lors de ce long processus ainsi que mes parents d'Amérique, Jacques et Louise. Je tiens à remercier mes quatre pattes, petites et grosses boules de poils, qui par leur joie de vivre, leur insouciance et leurs pitreries ont parfois allégé le poids du doctorat.

Merci également à mes amis qui m'ont aidé, soutenu, encouragé de toutes sortes de manières à aller jusqu'au bout.

Finalement, je remercie ma femme Claire pour son soutien constant et énergisant tout au long de mon doctorat. Merci du fond du cœur d'être présente à mes côtés et de me soutenir.

RÉSUMÉ

Cette thèse a pour objectif de formuler mathématiquement, d'analyser et d'implémenter deux méthodes sans factorisation pour l'optimisation non linéaire. Dans les problèmes de grande taille, la jacobienne des contraintes n'est souvent pas disponible sous forme de matrice ; seules son action et celle de sa transposée sur un vecteur le sont. L'optimisation sans factorisation consiste alors à utiliser des opérateurs linéaires abstraits représentant la jacobienne ou le hessien. De ce fait, seules les actions «opérateur-vecteur» sont autorisées et l'algèbre linéaire directe doit être remplacée par des méthodes itératives. Outre ces restrictions, une grande difficulté lors de l'introduction de méthodes sans factorisation dans des algorithmes d'optimisation concerne le contrôle de l'inexactitude de la résolution des systèmes linéaires. Il faut en effet s'assurer que la direction calculée est suffisamment précise pour garantir la convergence de l'algorithme concerné.

En premier lieu, nous décrivons l'implémentation sans factorisation d'une méthode de lagrangien augmenté pouvant utiliser des approximations quasi-Newton des dérivées secondes. Nous montrons aussi que notre approche parvient à résoudre des problèmes d'optimisation de structure avec des milliers de variables et contraintes alors que les méthodes avec factorisation échouent.

Afin d'obtenir une méthode possédant une convergence plus rapide, nous présentons ensuite un algorithme qui utilise un lagrangien augmenté proximal comme fonction de mérite et qui, asymptotiquement, se transforme en une méthode de programmation quadratique séquentielle stabilisée. L'utilisation d'approximations BFGS à mémoire limitée du hessien du lagrangien conduit à l'obtention de systèmes linéaires symétriques quasi-définis. Ceux-ci sont interprétés comme étant les conditions d'optimalité d'un problème aux moindres carrés linéaire, qui est résolu de manière inexacte par une méthode de Krylov. L'inexactitude de cette résolution est contrôlée par un critère d'arrêt facile à mettre en œuvre. Des tests numériques démontrent l'efficacité et la robustesse de notre méthode, qui se compare très favorablement à IPOPT, en particulier pour les problèmes dégénérés pour lesquels la LICQ n'est pas respectée à la solution ou lors de la minimisation.

Finalement, l'écosystème de développement d'algorithmes d'optimisation en Python, baptisé NLP.py, est exposé. Cet environnement s'adresse aussi bien aux chercheurs en optimisation qu'aux étudiants désireux de découvrir ou d'approfondir l'optimisation. NLP.py donne accès à un ensemble de blocs constituant les éléments les plus importants des méthodes d'optimisation continue. Grâce à ceux-ci, le chercheur est en mesure d'implémenter son algorithme en

se concentrant sur la logique de celui-ci plutôt que sur les subtilités techniques de son implémentation.

ABSTRACT

This thesis focuses on the mathematical formulation, analysis and implementation of two factorization-free methods for nonlinear constrained optimization. In large-scale optimization, the Jacobian of the constraints may not be available in matrix form; only its action and that of its transpose on a vector are. Factorization-free optimization employs abstract linear operators representing the Jacobian or Hessian matrices. Therefore, only operator-vector products are allowed and direct linear algebra is replaced by iterative methods. Besides these implementation restrictions, a difficulty inherent to methods without factorization in optimization algorithms is the control of the inaccuracy in linear system solves. Indeed we have to guarantee that the direction calculated is sufficiently accurate to ensure convergence.

We first describe a factorization-free implementation of a classical augmented Lagrangian method that may use quasi-Newton second derivatives approximations. This method is applied to problems with thousands of variables and constraints coming from aircraft structural design optimization, for which methods based on factorizations fail.

In order to obtain a method with a faster convergence rate, we present an algorithm that uses a proximal augmented Lagrangian as merit function and that asymptotically turns in a stabilized sequential quadratic programming method. The use of limited-memory BFGS approximations of the Hessian of the Lagrangian combined with regularization of the constraints leads to symmetric quasi-definite linear systems. Because such systems may be interpreted as the KKT conditions of linear least-squares problems, they can be efficiently solved using an appropriate Krylov method. Inaccuracy of their solutions is controlled by a stopping criterion which is easy to implement. Numerical tests demonstrate the effectiveness and robustness of our method, which compares very favorably with IPOPT, especially for degenerate problems for which LICQ is not satisfied at the optimal solution or during the minimization process.

Finally, an ecosystem for optimization algorithm development in Python, code-named NLP.py, is exposed. This environment is aimed at researchers in optimization and students eager to discover or strengthen their knowledge in optimization. NLP.py provides access to a set of building blocks constituting the most important elements of continuous optimization methods. With these blocks, users are able to implement their own algorithm focusing on the logic of the algorithm rather than on the technicalities of its implementation.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

CHAPITRE 1 INTRODUCTION

Au début des années 1960, Schmit introduit l'optimisation numérique à la communauté des ingénieurs aéronautiques en résolvant un problème d'optimisation structurelle à deux variables (Schmit, 1960). Ce fut la première fois qu'une méthode d'éléments finis était combinée à une méthode d'optimisation en un seul et unique programme. Depuis ce temps, des milliers d'articles de recherche ont été publiés sur ce sujet et les algorithmes d'optimisation ont été continuellement améliorés et raffinés, de même que les méthodes pour la formulation et la résolution de problèmes d'ingénierie.

De nos jours, les méthodes de points intérieurs et les méthodes de programmation quadratique séquentielle figurent parmi les méthodes d'optimisation les plus efficaces. Elles reposent toutes les deux sur la méthode de Newton et partagent ainsi les inconvénients habituels des méthodes de second ordre : à chaque itération, elles exigent la construction et la résolution de systèmes d'équations linéaires impliquant la matrice jacobienne des contraintes et/ou sa transposée. En pratique, les méthodes permettant de résoudre ces systèmes linéaires peuvent souffrir d'instabilités numériques causées, par exemple, par le mauvais conditionnement ou par la dégénérescence. En effet, lorsque les gradients des contraintes ne sont pas tous linéairement indépendants, le système linéaire utilisé pour chercher une nouvelle direction peut devenir singulier. De plus, avec l'augmentation de la taille et de la densité des systèmes, leur résolution à l'aide de l'algèbre linéaire directe devient prohibitive.

Cet aspect des méthodes du second ordre semble être un goulot d'étranglement pour leur développement et entrave leur capacité à résoudre d'énormes problèmes dont certains ne peuvent même pas être formulés explicitement. En effet, dans de nombreuses applications de l'ingénierie, la jacobienne des contraintes n'est pas disponible sous forme de matrice ; seules son action et celle de sa transposée sur un vecteur le sont. Ces constats motivent les travaux présentés dans cette thèse.

Afin de résoudre des problèmes de grande taille, nous tenons tout d'abord à éviter les besoins de stockage et de calcul excessifs qui pourraient nuire à leur résolution. Pour atteindre cette cible, nous imposons la condition que le hessien et la matrice jacobienne sont représentés par des opérateurs linéaires abstraits et que seules les actions «opérateur-vecteur» sont autorisées.

Sous ces conditions, il est évident que l'algèbre linéaire directe doit être remplacée par des méthodes itératives. L'introduction de méthodes sans factorisation dans des algorithmes d'optimisation crée une difficulté majeure. En effet, contrairement aux méthodes directes qui fournissent des directions très précises, les méthodes itératives calculent généralement des

directions inexactes. Cette inexactitude pose alors la question de savoir comment s'assurer que la direction calculée est suffisamment précise pour garantir la convergence de l'algorithme concerné. Pour les grands systèmes contenant des milliers d'équations, les méthodes itératives ont cependant des avantages décisifs par rapport aux méthodes directes en terme de vitesse et de demande mémoire. De plus, lorsque les exigences de précision sur le calcul de la direction recherchée ne sont pas trop importantes, seul un petit nombre d'itérations sera suffisant pour produire une direction acceptable.

L'idée d'une méthode d'optimisation sans factorisation n'est certainement pas nouvelle. Elle est présente dans la littérature d'optimisation depuis plusieurs années, avec l'apparition de la méthode de Newton inexacte (Dembo *et al.*, 1982) pour l'optimisation sans contrainte. Plus récemment, les travaux de Byrd *et al.* (2009); Curtis *et al.* (2010); Heinkenschloss et Ridzal (2014) décrivent de telles méthodes pour les problèmes avec contraintes d'égalités. Ces dernières sont toutefois rarement utilisées dans la pratique, et peu d'implémentations logicielles sont disponibles. Pour les problèmes contenant des inégalités, les méthodes de lagrangien augmenté sans factorisation telles que LANCELOT (Conn *et al.*, 1992) et ALGENCAN (Andreani *et al.*, 2008) sont les plus utilisées. Bien que LANCELOT permette l'utilisation de méthodes itératives n'impliquant que des produits jacobien-vecteur, son implémentation nécessite de former explicitement cette matrice et n'est alors pas complètement conforme aux exigences de la philosophie sans factorisation que nous nous sommes imposée. De plus, la séparabilité partielle par groupes, exploitée dans LANCELOT, rend le code extrêmement difficile à modifier. ALGENCAN, quant à lui, possède une option sans factorisation mais l'ajout des contraintes d'inégalité directement dans le lagrangien augmenté de Powell-Hestenes-Rockafellar (Rockafellar, 1973) conduit à des dérivées secondes discontinues. Finalement, ces deux logiciels sont écrits en Fortran et ne permettent pas facilement l'ajout de nouvelles fonctionnalités.

Compte tenu de ces obstacles, deux pistes principales de recherche s'imposent. Premièrement, le développement de méthodes sans factorisation pour l'optimisation non linéaire et deuxièmement la création d'un environnement facilitant la mise en œuvre efficace d'algorithmes d'optimisation.

La nécessité d'avoir des méthodes sans factorisation pour résoudre des problèmes de grande taille nous conduit à formuler et à implémenter deux méthodes sans factorisation : l'une étant une méthode de lagrangien augmenté et l'autre une méthode de lagrangien augmenté proximal se fondant naturellement en une méthode de programmation quadratique séquentielle. Nous avons naturellement décidé de commencer par développer un optimiseur sans factorisation basé sur une méthode de lagrangien augmenté. Bien que celle-ci ne converge pas aussi rapidement

que les algorithmes basés sur la méthode de Newton, elle est très populaire en optimisation et est conceptuellement plus simple à développer. En effet, celle-ci nécessite de résoudre de manière approximative une suite de sous-problèmes avec contraintes de bornes seulement, ce qui n'est pas le cas des méthodes basées sur la méthode de Newton pour qui les sous-problèmes contiennent des contraintes d'égalité et éventuellement de borne. Ainsi, l'adaptation d'une méthode de lagrangien augmenté à la philosophie sans factorisation énoncée précédemment semble plus facile que pour les méthodes de programmation quadratique séquentielle (SQP) ou pour celles des points intérieurs.

Malgré les résultats encourageants de cette première méthode, les algorithmes de lagrangien augmenté n'en restent pas moins des méthodes présentant une convergence plus lente que la programmation quadratique séquentielle ou celle des points intérieurs. La deuxième méthode proposée tente alors de palier à cet inconvénient en tirant partie de la convergence rapide des méthodes SQP.

Le deuxième axe de recherche porte sur le développement d'un environnement propice à la mise en œuvre d'algorithmes d'optimisation. Lors du développement d'un nouvel algorithme, les chercheurs en optimisation convoitent un environnement qui leur permettra de le mettre en œuvre rapidement et efficacement. De plus, un chercheur en optimisation doit pouvoir expérimenter facilement de nouvelles idées ou des variantes de méthodes connues en échangeant un détail d'implémentation pour un autre. Un exemple qui vient à l'esprit concerne la mise en œuvre d'un algorithme basé sur une recherche linéaire dans lequel celle-ci doit satisfaire les conditions fortes de Wolfe. Bien que nécessaire à la convergence, une telle recherche linéaire est beaucoup plus compliquée à mettre en œuvre et plus coûteuse qu'une simple recherche linéaire d'Armijo. Cependant, cette dernière ne satisfait éventuellement pas les hypothèses nécessaires à la convergence de l'algorithme, mais peut donner de bons résultats dans la pratique. Tout est alors question de trouver le bon équilibre entre la performance, le respect des exigences théoriques et la facilité de mise en œuvre.

Pour tenter de répondre à ces besoins, nous proposons dans cette thèse un environnement offrant la flexibilité, la puissance et la facilité d'utilisation pour l'enseignement de l'optimisation, l'étude de méthodes numériques, la recherche de nouveaux algorithmes et l'exploration de nouvelles idées, tout en conservant l'efficacité.

La présente thèse se divise en deux parties principales. Tout d'abord, le cadre théorique permet de recenser les écrits sur les notions et thèmes importants en lien avec notre sujet de recherche. Cette recension aide à dresser un portrait sommaire du sujet d'étude et à mieux comprendre la problématique de recherche. Cette revue de littérature est intentionnellement générique puisqu'une revue de littérature détaillée est fournie au début de chaque article. Après quelques

rappels de base portant sur l'optimisation, les conditions de qualifications de contraintes ainsi que sur les conditions d'optimalité, nous présentons dans le chapitre 2 les notions relatives aux méthodes locales, aux algorithmes de lagrangien augmenté, à la programmation quadratique séquentielle (SQP) et finalement aux méthodes de points intérieurs. Chacune de ces notions est introduite suivant son évolution historique et quelques uns des développements les plus récents sont mentionnés ainsi que les logiciels existants.

La seconde partie de cette thèse consiste en une succession d'articles constituant notre propre contribution. Le Chapitre 4 contient l'article « A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization », publié dans *Optimization and Engineering*. Le Chapitre 5 inclut l'article « A regularized factorization-free method for equality-constrained optimization » soumis dans *SIAM Journal on Optimization*. Le Chapitre 7 contient l'article « NLP.py: An Object-Oriented Environment for Large-Scale Optimization » soumis à *ACM Transactions on Mathematical Software (TOMS)*.

Nous résumons les contributions de cette thèse et discutons des perspectives de ce travail au Chapitre 8. Enfin, le Chapitre 9 conclut cette thèse.

CHAPITRE 2 REVUE DE LITTÉRATURE

2.1 Introduction

L'optimisation réfère à l'étude et à la recherche de la valeur optimale d'une fonction donnée considérant un ensemble de contraintes ou restrictions appliquées sur celle-ci. De nos jours, l'optimisation intervient dans un grand nombre d'applications telles que les télécommunications, la planification de traitements de radiothérapie, le dimensionnement de structures, la bourse et l'aérodynamique, pour n'en mentionner que quelques unes.

Un problème d'optimisation est un problème mathématique de la forme

$$\min_{x \in \Omega} \quad f(x), \tag{2.1}$$

où $x \in \mathbb{R}^n$ est le vecteur des variables, $f : \mathbb{R}^n \to \mathbb{R}$ est la fonction objectif et Ω, appelée la région admissible pour x, est un ensemble de restrictions imposées sur les variables x. L'optimisation peut se diviser en deux domaines.

Tout d'abord, la modélisation consiste à choisir une fonction f et un ensemble Ω qui représentent un problème concret. L'étape de modélisation en sélectionne les caractéristiques principales dans le but d'élaborer un modèle mathématique (2.1) fidèle afin, dans un second temps, de l'optimiser par un algorithme d'optimisation. Un grand soin doit être porté à la modélisation puisque les choix posés lors de cette étape influeront indéniablement la difficulté à résoudre ce problème.

En second lieu, il s'agit de se pencher sur la résolution du modèle mathématique (2.1) et sur l'évaluation de la qualité de la solution obtenue. Les choix effectués lors de la modélisation déterminent les familles d'algorithmes qui seront adéquates pour résoudre le problème (2.1). Les chercheurs ont divisé le problème (2.1) en plusieurs sous-catégories définies par les caractéristiques de f et/ou Ω. Par exemple, la fonction objectif pourrait être linéaire, quadratique ou convexe, tandis que les variables, elles, pourraient être continues ou discrètes. Classifier les problèmes d'optimisation aide à faire un choix éclairé quand vient le temps de choisir un algorithme ou un solveur. En règle générale, aucun algorithme ou solveur n'est le meilleur pour tous les types de problèmes, il doit être choisi attentivement.

Dans cette thèse, nous nous concentrons sur les méthodes d'optimisation traitant les problèmes

d'optimisation généraux de la forme

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.c.} & h(x) = 0, \\ & c(x) \geq 0 \end{array} \tag{NLP}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$ et $c : \mathbb{R}^n \to \mathbb{R}^m$ sont des fonctions de classe $\mathcal{C}^2$ éventuellement non linéaires et non-convexes. Notons $\Omega := \{x \in \mathbb{R}^n \mid h(x) = 0 \text{ et } c(x) \geq 0\}$ le domaine admissible de (NLP).

Pour le problème (NLP), on dit que $x^* \in \Omega$ est

- un minimum global de f si et seulement si pour tout $x \in \Omega$, $f(x) \geq f(x^*)$;
- un minimum local de f si et seulement s'il existe une boule centrée en x^* de rayon $\epsilon > 0$ telle que pour tout x dans cette boule et dans Ω, $f(x) \geq f(x^*)$;
- un minimum local strict de f si et seulement s'il existe une boule centrée en x^* de rayon $\epsilon > 0$ telle que pour tout $x \neq x^*$ dans cette boule et dans Ω, $f(x) > f(x^*)$.

Après quelques rappels sur les conditions de qualifications de contraintes ainsi que sur les conditions d'optimalité visant à identifier un minimum, nous présentons les grandes familles de méthodes numériques permettant de résoudre un problème d'optimisation : les algorithmes de lagrangien augmenté, de programmation quadratique séquentielle (SQP) et finalement les méthodes de points intérieurs. Chacune de ces notions est introduite suivant son évolution historique et quelques uns des développements les plus récents sont mentionnés ainsi que les logiciels existants. La revue de littérature qui suit est intentionnellement assez générale puisqu'une revue de littérature détaillée s'appliquant spécifiquement aux problématiques traitées dans chaque article est fournie au début de chacun d'eux.

2.2 Qualification des contraintes et conditions d'optimalité

La satisfaction des conditions d'optimalité du problème (NLP) est l'une des bases de la programmation non linéaire. Les conditions d'optimalité représentent une caractérisation explicite du fait que $f(x) \geq f(x^*)$, globalement ou localement. Sous certaines hypothèses idéales sur le problème (NLP), par exemple, la convexité, ces conditions sont à la fois nécessaires et suffisantes. Dans le cas où ces hypothèses idéales ne sont pas vérifiées, la satisfaction d'une condition de qualification de contraintes assure que les conditions d'optimalité du premier ordre sont des conditions nécessaires d'optimalité. La recherche des points critiques de (NLP) peut alors être effectuée parmi les points qui satisfont les conditions d'optimalité du premier ordre.

Les conditions de qualification de contraintes constituent un ensemble d'hypothèses qui assurent que l'ensemble des contraintes Ω et sa linéarisation autour d'un certain point coïncident et constituent des hypothèses suffisantes à l'existence des multiplicateurs de Lagrange. Ces conditions impliquent généralement les gradients des contraintes actives.

Définition 1. *On dit que la contrainte c_j (pour $j \in \{1, ..., m\}$) est active en $x^* \in \Omega$ si $c_j(x^*) = 0$.*
L'ensemble des contraintes actives en x^* est noté $\mathcal{A}(x^*) = \{j \in \{1, ..., m\} \mid c_j(x^*) = 0\}$.

Dans la littérature, de nombreuses conditions de qualification de contraintes ont été énoncées. Wang *et al.* (2013) recensent la majorité des conditions de qualification de contraintes connues à la date de parution de l'article pour l'optimisation non linéaire lisse en dimension finie et décrivent les liens qui unissent celles-ci. Peu d'entre elles sont cependant vérifiables en pratique. La condition suivante, facilement vérifiable en pratique, est sans doute la plus connue de toutes les conditions de qualification.

Définition 2. *Soit $x^* \in \Omega$, on dit que la condition de qualification d'indépendance linéaire (LICQ) est satisfaite si*

$$\{\nabla c_i(x^*) \mid i \in \mathcal{A}(x^*)\} \cup \{\nabla h_i(x^*) \mid i = 1, ..., p\}$$

est un ensemble de vecteurs linéairement indépendants de $\mathbb{R}^n$.

En d'autres mots, si la LICQ est satisfaite en x^*, alors tous les gradients actifs sont linéairement indépendants.

Dans le cas où le problème (NLP) ne possède que des égalités, la MFCQ (Mangasarian-Fromovitz Constraint Qualification) et la LICQ sont équivalentes.

Énonçons maintenant les conditions d'optimalité de premier ordre pour le problème (NLP).

Théorème 1. *Considérons le problème* (NLP). *Supposons que la condition de qualification LICQ soit satisfaite en $x^* \in \Omega$. Si x^* est un minimum local, alors il existe $\delta \in \mathbb{R}^m$ et $\lambda \in \mathbb{R}^p$ tel que :*

$$\nabla f(x^*) - \sum_{i=1}^{p} \lambda_i \nabla h_i(x^*) - \sum_{i=1}^{m} \delta_i \nabla c_i(x^*) = 0, \tag{2.2}$$

$$\delta_i c_i(x^*) = 0, \quad i = 1, ..., m, \tag{2.3}$$

$$h(x^*) = 0, \tag{2.4}$$

$$c(x^*) \geq 0, \tag{2.5}$$

$$\delta \geq 0. \tag{2.6}$$

où δ et λ sont appelés les multiplicateurs de Lagrange. Ces conditions sont aussi connues sous le nom de conditions de Karush, Kuhn et Tucker (KKT). La condition (2.3), appelée condition de complémentarité, joue un rôle important lors de la résolution de (NLP). Nous dirons que la condition de complémentarité stricte est satisfaite lorsque

$$\delta_i^* > 0 \Leftrightarrow c_i(x^*) = 0 \quad \forall\ i = 1, ..., m. \tag{2.7}$$

En d'autres mots, si une contrainte est active, alors le multiplicateur associé ne peut être que nul. Cette condition sera utilisée lorsque nous aborderons les méthodes de points intérieurs dans la section 2.4.3. Elle joue un rôle primordial car, combinée à la LICQ, elle garantit que la matrice jacobienne du système de KKT est inversible à la solution.

En définissant le lagrangien de (NLP) par

$$\mathcal{L}(x; \delta, \lambda) := f(x) - \lambda^T h(x) - \delta^T c(x), \tag{2.8}$$

les équations (2.2) des conditions de KKT sont équivalentes à $\nabla_x \mathcal{L}(x^*; \delta, \lambda) = 0$. Ainsi un point x^* qui satisfait les conditions de KKT (2.2)-(2.6) est un point stationnaire pour la fonction lagrangienne respectivement aux variables x.

Le théorème suivant, de Gauvin (1977), montre l'importance de la MFCQ dans les conditions de KKT. Il donne des conditions nécessaires et suffisantes d'existence d'un ensemble de multiplicateurs de Lagrange non vide et compact.

Théorème 2. *Soit x^* un minimum local de* (NLP). *Si $\Gamma(x^*)$, l'ensemble des multiplicateurs de Lagrange associés à x^* défini par*

$$\Gamma(x^*) := \{(\delta^*, \lambda^*) \in \mathbb{R}^m \times \mathbb{R}^p \mid (x^*, \delta^*, \lambda^*) \textit{ vérifie } (2.2)\text{-}(2.6)\},$$

alors

$$\Gamma(x^*) \neq \emptyset \textit{ et est borné } \Leftrightarrow \textit{la MFCQ est vérifiée en } x^*.$$

Ainsi, si la MFCQ n'est pas satisfaite en x^*, il se peut qu'il n'existe pas de multiplicateurs satisfaisants les conditions de KKT ou même qu'il existe un ensemble non borné de multiplicateurs comme dans le cas des problèmes avec contraintes d'équilibre (Leyffer, 2004). Dans ces deux cas, (NLP) est dit dégénéré et des difficultés numériques peuvent survenir dans les méthodes numériques permettant de le résoudre. Nous reparlerons de cela dans les prochaines sections.

Définition 3. *Soient* $(x^*, \delta^*, \lambda^*)$ *un point de KKT satisfaisant les conditions* (2.2)-(2.6)*, et* $\mathcal{A}(x^*)$*. L'ensemble des contraintes actives en* x^**, l'ensemble*

$$\mathcal{K}(x^*, \delta^*, \lambda^*) = \left\{ d \in \mathbb{R}^n \;\middle|\; \begin{array}{ll} \nabla h_i(x^*)^T d = 0, & i = 1, ..., p, \\ \nabla c_i(x^*)^T d = 0, & j \in \mathcal{A}(x^*) \text{ tel que } \delta_i > 0, \\ \nabla c_i(x^*)^T d \geq 0, & j \in \mathcal{A}(x^*) \text{ tel que } \delta_i = 0. \end{array} \right\}$$

est appelé cône critique en $(x^*, \delta^*, \lambda^*)$*.*

Ce cône joue un rôle prépondérant dans la recherche d'un minimum local de (NLP) comme le montre le théorème suivant donnant un ensemble de conditions sur la matrice hessienne du lagrangien.

Théorème 3. *Soient* $(x^*, \delta^*, \lambda^*)$ *un point de KKT satisfaisant les conditions* (2.2)-(2.6) *et où la LICQ est vérifiée, alors*

$$d^T \nabla^2_{xx} \mathcal{L}(x^*; \delta^*, \lambda^*) d \geq 0, \quad \forall d \in \mathcal{K}(x^*, \delta^*, \lambda^*).$$

Ces conditions sont appelées conditions nécessaires du second ordre. Le résultat suivant apporte quant à lui des conditions suffisantes afin d'identifier un minimum local strict de (NLP).

Théorème 4. *Soient* $(x^*, \delta^*, \lambda^*)$ *satisfaisant les conditions de KKT* (2.2)-(2.6)*. Si*

$$d^T \nabla^2_{xx} \mathcal{L}(x^*; \delta^*, \lambda^*) d > 0, \quad \forall d \neq 0 \text{ tel que } d \in \mathcal{K}(x^*, \delta^*, \lambda^*),$$

alors x^* *est un minimum local strict de* (NLP)*.*

Autrement dit, le hessien du lagrangien est défini positif sur le cône tangent aux contraintes actives représenté par $\mathcal{K}(x^*, \delta^*, \lambda^*)$.

2.3 Méthodes numériques

2.3.1 L'optimisation sans contrainte

Soit le programme mathématique sans contrainte,

$$\min_{x \in \mathbb{R}^n} \quad f(x), \tag{2.9}$$

dans lequel $f : \mathbb{R}^n \to \mathbb{R}$. Dans cette section, nous revoyons quelques méthodes locales d'optimisation qui permettent d'identifier un point stationnaire de ce programme mathématique. Dans cette section, les conditions de KKT (2.2)-(2.6) se résument à $\nabla f(x) = 0$.

2.3.2 La méthode de la plus forte pente

La plus ancienne des méthodes pour résoudre (2.9), et sûrement la plus simple, est la méthode de la plus forte pente. Afin d'identifier une solution de (2.9), cette méthode, comme toutes les méthodes de descente, identifie un pas p_k pour lequel une décroissance de f est attendue. Ensuite, l'itéré courant x_k est mis à jour par la règle $x_{k+1} = x_k + p_k$. Le pas p_k est obtenu en prenant l'opposé de la direction du gradient en x_k

$$p_k = -\nabla f(x_k).$$

Bien que le pas p_k soit toujours une direction de descente pour $\nabla f(x_k) \neq 0$, et que cette méthode soit globalement convergente, lorsque combinée à une recherche linéaire, elle présente une faible vitesse de convergence et montre un comportement en zigzag (Nocedal *et al.*, 2002). Malgré ces inconvénients, elle reste très populaire d'une part parce qu'elle ne requiert que l'accès aux dérivées premières de f et d'autre part puisqu'elle est extrêmement simple à mettre en œuvre.

2.3.3 La méthode de Newton

Lorsque les dérivées secondes sont disponibles, la méthode locale la plus connue est la méthode de Newton pour laquelle un pas p_k est identifié en résolvant le système

$$\nabla^2 f(x_k) p_k = -\nabla f(x_k). \tag{2.10}$$

Cette méthode construit une approximation quadratique de f autour du point x_k

$$q(x_k + p_k) = f(x_k) + \nabla f(x_k)^T p_k + \tfrac{1}{2} p_k^T \nabla^2 f(x_k) p_k$$

et en cherche un point stationnaire. Si la matrice hessienne de f en x_k est définie positive, alors le pas $p_k^N = -\nabla^2 f(x_k)^{-1} \nabla f(x_k)$ est le minimum de ce modèle quadratique et celui-ci est une direction de descente pour f en x_k. Si elle n'est pas définie positive, alors une solution de (2.10) n'est pas nécessairement une direction de descente. Pour remédier à ce problème, de petites modifications de la matrice hessienne ou du pas p_k^N sont apportées afin d'obtenir une direction de descente pour f, voir, par exemple, (Nocedal et Wright, 2006, Chapitre 3).

L'un des atouts majeurs de cette méthode réside dans sa vitesse de convergence : lorsque la méthode de Newton est démarrée suffisamment proche de x^*, elle converge quadratiquement.

2.3.4 Les méthodes quasi-Newton

Les méthodes quasi-Newton sont une alternative à la méthode de Newton, lorsque le calcul des dérivées secondes n'est pas possible, pas souhaitable ou lorsque la résolution du système linéaire (2.10) n'est pas envisageable. Les méthodes quasi-Newton ont conduit à de grandes améliorations par rapport à la méthode de la plus forte pente et elles se montrent parfois plus efficaces que la méthode de Newton, bien qu'elles ne demandent pas le calcul des dérivées secondes. Le changement dans le gradient à deux itérations successives sert à construire une approximation du hessien de la fonction objectif qui, sous certaines conditions, produira un algorithme ayant une convergence superlinéaire.

La première méthode quasi-Newton, suggérée par Davidon, date de la fin des années 50 mais fut seulement publiée en 1991 (Davidon, 1991). Au vu des travaux de Davidon, Fletcher et Powell (1963) lancent le début de recherches intensives sur les méthodes quasi-Newton. Celles-ci comprennent une classe de méthodes qui imitent la méthode de Newton en utilisant une approximation symétrique B_k du hessien exact $\nabla^2 f(x_k)$. Dans ces méthodes, un pas p_k est généré en résolvant

$$B_k p_k = -\nabla f(x_k).$$

Lorsque B_k est définie positive, la direction $p_k^{QN} = -B_k^{-1}\nabla f(x_k)$ est une direction de descente pour f en x_k. Dans la littérature, on retrouve souvent la notation H_k pour désigner l'inverse de B_k.

La matrice B_k est actualisée à chaque itération par une formule de mise à jour dite quasi-Newton qui tient compte de l'information nouvellement disponible. D'après le théorème de Taylor, la matrice hessienne $\nabla^2 f(x_k)$ vérifie

$$\nabla^2 f(x_k)(x_{k+1} - x_k) \approx \nabla f(x_{k+1}) - \nabla f(x_k).$$

La mise à jour quasi-Newton cherche à imiter ce comportement lors de la construction de la nouvelle matrice B_{k+1}, qui elle, doit satisfaire l'équation sécante

$$B_{k+1} s_k = y_k$$

où $s_k := x_{k+1} - x_k$ et $y_k := \nabla f(x_{k+1}) - \nabla f(x_k)$.

Une classe de mise à jour très connue pour les méthodes quasi-Newton est la famille Broyden

à un paramètre (Broyden, 1970). Cette famille dépend du paramètre ϕ_k, appelé paramètre de Broyden. Elle inclut la méthode DFP (Davidon-Fletcher-Powell) lorsque ϕ_k vaut 1, qui fut la première méthode quasi-Newton suggérée dans (Davidon, 1991; Fletcher et Powell, 1963). Elle contient aussi la méthode SR1 (Symmetric Rank One) et, la plus connue, la méthode BFGS découverte indépendamment par Broyden, Fletcher, Goldfarb et Shanno, pour laquelle ϕ_k vaut 0. Toutes les méthodes de la famille Broyden sont des mises à jour de rang 2 sauf SR1 qui, comme son nom l'indique, est de rang 1. Ces approximations possèdent une propriété de symétrie héréditaire, c'est-à-dire que si B_k est initialisée avec une matrice symétrique, alors la mise à jour garantit que la symétrie se transmet à la matrice suivante.

Lorsque la condition de courbure $s_k^T y_k > 0$ est satisfaite et que les méthodes BFGS et DFP sont initialisées avec des matrices définies positives, les matrices B_k sont toujours définies positives. C'est un avantage que nous utiliserons dans le chapitre 5. Toutefois, ce n'est pas nécessairement le cas pour la méthode SR1. Conn *et al.* (1988) constatent que les matrices générées par la mise à jour SR1 semblent fournir de meilleures approximations du hessien exact par rapport à celles produites par DFP ou BFGS. Cependant elle est sujette à des instabilités numériques et parfois mène à des échecs. Ainsi, elle n'est que peu utilisée en pratique.

Intuitivement, il semble naturel de penser que la convergence rapide de la méthode de Newton sera conservée pour les méthodes quasi-Newton si les matrices B_k convergent vers le hessien exact $\nabla^2 f(x^*)$. En fait, elle sera conservée sous des hypothèses beaucoup plus faibles : B_k doit converger suffisamment vite vers $\nabla^2 f(x^*)$ uniquement dans la direction du pas p_k, i.e.

$$\lim_{k \to +\infty} \frac{\|(B_k - \nabla^2 f(x^*))p_k\|}{\|p_k\|} = 0.$$

Cette condition, appelée condition de Dennis et Moré (Dennis et Moré, 1974), est fondamentale pour les méthodes quasi-Newton car elle donne une condition nécessaire et suffisante pour obtenir une convergence locale rapide de celles-ci.

Dans la pratique, on utilise en général une variante à mémoire limitée des méthodes présentées ci-dessus. Des approximations implicites du hessien sont maintenues en ne gardant en mémoire que quelques vecteurs de taille n au lieu de matrices denses $n \times n$. Toutes les méthodes quasi-Newton précédentes possèdent des variantes à mémoire limitée. Dans le cadre de cette thèse, uniquement les versions à mémoire limitée de BFGS (L-BFGS) et SR1 (L-SR1) ont été utilisées. L'idée principale de ces méthodes consiste à se servir de l'information sur la courbure des m plus récentes itérations afin de construire une approximation du hessien. L'information sur la courbure des itérations plus anciennes, qui est susceptible d'être désuète et de ne pas

représenter convenablement le hessien à l'itération courante, est supprimée afin d'économiser de l'espace de stockage. Ainsi, uniquement les m paires de vecteurs les plus récentes (s_k, y_k) sont conservées. Les produits hessien-vecteur $B_k v$ peuvent alors être obtenus en effectuant une suite de produits scalaires et d'additions vectorielles impliquant uniquement v et les paires (s_i, y_i) pour $i = m - k + 1, ..., k$. Lors d'expérimentations numériques, un nombre faible de paires stockées m (entre 1 et 7) donne des résultats généralement satisfaisants.

2.4 L'optimisation avec contraintes

Considérons le programme mathématique non linéaire suivant

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{s.c. } h(x) = 0, \; \ell \leq x \leq u. \tag{2.11}$$

Cette formulation, équivalente à (NLP), suppose que toutes les contraintes générales d'inégalité $c(x) \geq 0$ ont été converties préalablement en équations par l'introduction de variables d'écart. Dans cette section, nous présentons les différents types de méthodes utilisées pour résoudre numériquement (2.11).

2.4.1 Les méthodes de lagrangien augmenté

Les méthodes de lagrangien augmenté ont été popularisées par l'apparition du logiciel LANCELOT dans les années 1990 (Conn *et al.*, 1992) et restent encore actuellement très utilisées. Plusieurs livres entiers sont dédiés à ces méthodes : Bertsekas (1982), Conn *et al.* (2000) et Birgin et Martínez (2014). Nocedal et Wright (2006, Chapitre 17) proposent, quant à eux, un aperçu plus général sur ces méthodes.

La méthode du lagrangien augmenté pour les problèmes d'égalité apparait à la fin des années 1960 dans les travaux de Hestenes (1969) et Powell (1969) pour faire face aux difficultés dues au mauvais conditionnement associé à la méthode de pénalisation quadratique. Au début des années 70, Rockafellar (1973) et Buys (1972) ont étendu le lagrangien augmenté de Hestenes-Powell au cas des problèmes d'optimisation avec contraintes d'inégalité.

Pour le problème (2.11), on définit le lagrangien par

$$\mathcal{L}(x, \lambda) := f(x) + h(x)^T \lambda \tag{2.12}$$

où λ est une estimation des multiplicateurs de Lagrange associés à h et on définit le lagrangien

augmenté de Powell-Hestenes-Rockafellar par

$$\phi(x; \lambda, \rho) := \mathcal{L}(x, \lambda) + \tfrac{1}{2}\rho\|h(x)\|_2^2 \tag{2.13}$$

où $\rho > 0$ est le paramètre de pénalisation.

Une méthode de lagrangien augmenté consiste à résoudre une suite de sous-problèmes ne possédant que des contraintes de borne

$$\min_{x \in \mathbb{R}^n} \ \phi(x; \lambda_k, \rho_k) \quad \text{s.c. } \ell \leq x \leq u, \tag{2.14}$$

pour des valeurs fixées de $\lambda_k \approx \lambda^*$ et ρ_k. Chaque résolution du sous-problème est suivie de mises à jour des multiplicateurs λ_k et du paramètre de pénalité ρ_k. Ces mises à jour sont généralement basées sur la réduction de la violation des contraintes obtenue dans le plus récent sous-problème. Si la réduction est suffisante, alors une nouvelle estimation des multiplicateurs λ_k est calculée et ρ_k ne change pas. Par contre, lorsque la réduction n'est pas suffisante, ρ_k est augmenté afin de forcer une décroissance de la violation des contraintes et les multiplicateurs restent identiques (Nocedal et Wright, 2006, Algorithm 17.4). La résolution des sous-problèmes joue un rôle clef dans l'efficacité des méthodes de lagrangien augmenté. Ainsi, un soin particulier doit y être apporté.

La méthode du lagrangien augmenté, contrairement à la méthode de pénalisation quadratique, converge sans que le paramètre de pénalité ρ_k ne tende vers l'infini et par conséquent souffre moins de mauvais conditionnement. Cependant, d'importants inconvénients sont imputables à ces méthodes. En particulier, un mauvais choix des valeurs initiales du paramètre de pénalité et des multiplicateurs de Lagrange (Curtis *et al.*, 2014).

Les propriétés de convergence locale des méthodes de lagrangien augmenté ont été discutées dans (Bertsekas, 1976; Goldfarb *et al.*, 1999; Birgin *et al.*, 2012). Bertsekas (1976) montre que, si la valeur initiale des multiplicateurs de Lagrange est suffisamment proche de la valeur optimale, la convergence de la méthode de lagrangien augmenté est linéaire avec un taux de convergence inversement proportionnel au paramètre de pénalité. Pour établir ce résultat, une résolution exacte de chaque sous-problème (2.14) a été supposée (Bertsekas, 1973).

ALGENCAN (Andreani *et al.*, 2008; Birgin et Martínez, 2014), LANCELOT (Conn *et al.*, 1992), MINOS (Murtagh et Saunders, 2003) et PENNON (Kočvara et Stingl, 2011) comptent parmi les implémentations les plus efficaces des méthodes de lagrangien augmenté pour les problèmes de grande taille.

ALGENCAN et LANCELOT résolvent tous les deux une suite de sous-problèmes avec des

contraintes de borne dont la fonction objectif est le lagrangien augmenté de Powell-Hestenes-Rockafellar. Ils diffèrent notamment dans le choix de leur technique de globalisation ainsi que dans la façon de résoudre approximativement les sous-problèmes. LANCELOT utilise une méthode de région de confiance tandis qu'ALGENCAN utilise soit une stratégie de recherche linéaire soit une méthode de région de confiance. En outre, LANCELOT a été conçu pour tirer parti de la structure partiellement séparable des fonctions objectif et contrainte dans l'évaluation du hessien du lagrangien ou même dans les approximations quasi-Newton de celui-ci. MINOS, quant à lui, résout une suite de sous-problèmes avec contraintes dans lesquels chaque sous-problème consiste à minimiser le lagrangien augmenté sujet à une linéarisation des contraintes.

Finalement, PENNON est un code plus général qui traite aussi des problèmes d'optimisation non linéaire semi définie positive. Sa grande différence par rapport aux trois méthodes précédentes réside dans la définition de sa fonction lagrangien augmenté. Celle-ci est une combinaison du lagrangien augmenté classique pour les contraintes d'égalité et d'une fonction barrière pour les contraintes d'inégalité.

2.4.2 Les méthodes SQP

La méthode de programmation quadratique séquentielle (SQP) est l'une des méthodes les plus efficaces pour résoudre les problèmes d'optimisation non linéaires avec contraintes. Les prémisses de cette méthode ont été jetées dans la thèse de Wilson (1963). Les méthodes SQP furent popularisées à la fin des années 1970 par les articles de Han (1977) et Powell (1978). L'attrait pour ces méthodes provient de leurs propriétés de convergence rapide, exposées dans des expérimentations numériques par Schittkowski (1982). Un traitement complet de l'histoire, de la théorie et de l'implémentation pratique de ces méthodes est donné dans Boggs et Tolle (1995) et Gould et Toint (2000). De nombreux livres d'optimisation contiennent également des chapitres entiers sur ces méthodes, par exemple Nocedal et Wright (2006, Chapitre 18).

Une méthode SQP consiste à minimiser une suite de modèles quadratiques de la fonction objectif ou du lagrangien sujet à une linéarisation des contraintes autour de l'itéré courant. Pour la simplicité de l'exposé et en prévision de l'article 2 présenté au Chapitre 5, nous développons la théorie des méthodes SQP pour les problèmes ne comportant que des égalités :

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.c.} & h(x) = 0 \end{array} \tag{2.15}$$

où $f : \mathbb{R}^n \to \mathbb{R}$, $h : \mathbb{R}^n \to \mathbb{R}^p$ sont des fonctions de classe $\mathcal{C}^2$.

Commençons par une présentation de la méthode SQP locale qui constitue un excellent algorithme pour résoudre (2.15) lorsque le point de départ est suffisamment proche de la solution optimale. Une approche habituelle pour formuler un algorithme d'optimisation est d'utiliser les conditions d'optimalité du premier ordre du problème (2.15) pour définir un système d'équations non linéaires à résoudre. En effet, tout minimum local de (2.15) qui vérifie une condition de qualification est solution du système :

$$\nabla f(x) - J(x)^T \lambda = 0, \tag{2.16}$$

$$h(x) = 0, \tag{2.17}$$

où $J(x)$ est la matrice jacobienne des contraintes au point x et $\lambda \in \mathbb{R}^p$ sont les multiplicateurs de Lagrange associés aux égalités h. Ces équations peuvent être résolues efficacement en utilisant la méthode de Newton. Considérons une itération de celle-ci, à partir des estimations courantes x_k et λ_k des variables primales et duales. Les équations de Newton s'écrivent alors

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k;\lambda_k) & J(x_k)^T \\ J(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) - J(x_k)^T\lambda_k \\ h(x_k) \end{bmatrix}. \tag{2.18}$$

Finalement, le nouvel itéré (x_{k+1}, λ_{k+1}) est obtenu en posant

$$x_{k+1} = x_k + \Delta x \tag{2.19}$$

$$\lambda_{k+1} = \lambda_k + \Delta\lambda. \tag{2.20}$$

Mais alors d'où vient le mot « quadratique » dans les méthodes SQP ? En regardant attentivement les équations de Newton (2.18), on remarque qu'elles représentent les conditions d'optimalité de premier ordre du problème quadratique

$$\begin{array}{ll} \min\limits_{\Delta x} & \nabla\mathcal{L}(x_k;\lambda_k)^T\Delta x + \frac{1}{2}\Delta x^T \nabla^2_{xx}\mathcal{L}(x_k;\lambda_k)\Delta x \\ \text{s.c.} & J(x_k)\Delta x = -h(x_k), \end{array} \tag{2.21}$$

où $\Delta\lambda$ sont les multiplicateurs de Lagrange associés aux contraintes d'égalité de (2.21).

Lorsque écrit en terme des variables x, ce problème quadratique devient

$$\begin{array}{ll} \min\limits_{x} & \nabla\mathcal{L}(x_k;\lambda_k)^T(x - x_k) + \frac{1}{2}(x - x_k)^T \nabla^2_{xx}\mathcal{L}(x_k;\lambda_k)(x - x_k) \\ \text{s.c.} & J(x_k)(x - x_k) = -h(x_k). \end{array} \tag{2.22}$$

Une extension simpliste de la méthode SQP pour les problèmes possédant des inégalités (2.11)

consisterait à résoudre une suite de sous-problèmes quadratiques auxquels nous ajoutons les contraintes de borne

$$\begin{array}{ll} \min\limits_{x} & \nabla\mathcal{L}(x_k;\lambda_k)^T(x-x_k)+\frac{1}{2}(x-x_k)^T\nabla^2_{xx}\mathcal{L}(x_k;\lambda_k)(x-x_k) \\ \text{s.c.} & J(x_k)(x-x_k)=-h(x_k) \quad \text{et} \quad \ell \leq x \leq u. \end{array} \tag{2.23}$$

La méthode SQP locale, que nous venons de présenter, consiste essentiellement à appliquer la méthode de Newton aux conditions de KKT de (2.15). Ainsi, leur mise en œuvre doit comprendre un mécanisme de globalisation, car cette méthode n'est généralement pas globalement convergente. Deux techniques de globalisation sont traditionnellement employées : les méthodes de recherche linéaire et les régions de confiance.

Un prototype de méthode SQP employant une recherche linéaire commence par calculer un pas Δx, solution du système linéaire (2.18). Le prochain itéré est obtenu par $x_{k+1} = x_k + \alpha_k \Delta x$ où la longueur du pas α_k est trouvée en exigeant que $\phi(x_k + \alpha_k \Delta x)$ soit suffisamment plus petite que $\phi(x_k)$ pour une certaine fonction de mérite ϕ. Une recherche linéaire d'Armijo commençant par le pas unité identifierait une telle longueur du pas α_k. Des méthodes plus sophistiquées assurant une décroissance suffisante de ϕ sont présentées, par exemple, dans Ortega et Rheinboldt (2000) et Moré et Thuente (1994). L'algorithme 2.4.1 résume les étapes importantes d'une méthode SQP avec recherche linéaire. Lorsque les dérivées secondes du lagrangien ne sont pas disponibles ou sont trop coûteuses à évaluer, le hessien du lagrangien $\nabla^2_{xx}\mathcal{L}(x_k;\lambda_k)$, nécessaire à l'étape 3 de cet algorithme, peut être remplacé par une approximation H_k, généralement symétrique. Les nombreuses méthodes SQP avec recherche linéaire proposées dans la littérature diffèrent principalement dans leur choix de cette matrice H_k et de la fonction de mérite ϕ.

Algorithm 2.4.1 Méthode SQP avec recherche linéaire

1: Choisir un point initial (x_0, λ_0) et initialiser le compteur d'itération $k = 0$.
2: Si un certain critère de convergence est atteint, alors STOP.
3: Choisir $H_k \approx \nabla^2_{xx}\mathcal{L}(x_k;\lambda_k)$ et résoudre le système linéaire

$$\begin{bmatrix} H_k & J(x_k)^T \\ J(x_k) & 0 \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) - J(x_k)^T\lambda_k \\ h(x_k) \end{bmatrix} \tag{2.24}$$

pour obtenir un pas $(\Delta x, \Delta\lambda)$.
4: Choisir $\alpha_k > 0$ telle que $\phi(x_k + \alpha_k \Delta x)$ est suffisamment plus petite que $\phi(x_k)$.
5: Poser $x_{k+1} = x_k + \alpha_k \Delta x$ et $\lambda_{k+1} = \lambda_k + \Delta\lambda$.
6: Mettre à jour le compteur des itérations $k \leftarrow k + 1$ et retourner à l'étape 2.

Les méthodes SQP avec région de confiance imposent une restriction supplémentaire au

sous-problème quadratique (2.21) en ajoutant une région de confiance. Ainsi, le sous-problème à résoudre prend la forme

$$\begin{array}{ll} \min\limits_{\Delta x} & \nabla\mathcal{L}(x_k;\delta_k)^T\Delta x + \frac{1}{2}\Delta x^T\nabla^2_{xx}\mathcal{L}(x_k;\delta_k)\Delta x \\ \text{s.c.} & J(x_k)\Delta x = -h(x_k), \\ & \|\Delta x\| \leq \Delta_k \end{array} \tag{2.25}$$

pour un certain rayon $\Delta_k > 0$. Si la valeur de la fonction de mérite s'accorde avec un modèle de la fonction de mérite en ce point, alors le pas Δx est accepté et le nouvel itéré est obtenu en posant $x_{k+1} = x_k + \Delta x$. λ_{k+1} est, quant à lui, obtenu en minimisant $\|\nabla f(x_{k+1}) - J(x_{k+1})^T\lambda\|$. De plus, le rayon de la région de confiance est élargi. Lorsque la fonction de mérite ne s'accorde pas avec le modèle, alors le pas est rejeté ($x_{k+1} = x_k$ et $\lambda_{k+1} = \lambda_k$) et le rayon est diminué. Le chapitre 15 de Conn *et al.* (2000) couvre en détails quelques unes des méthodes SQP avec région de confiance les plus connues.

Lorsque les dérivées secondes ne sont pas accessibles, Powell (1978) fut le premier à suggérer d'utiliser une approximation H_k du hessien du lagrangien, symétrique, définie positive telle que BFGS. Cela peut sembler être un choix surprenant étant donné que le hessien du lagrangien est généralement indéfini à la solution. Cependant, la proposition de Powell est basée sur l'observation que la courbure approximative est susceptible d'être positive proche d'un minimum local isolé, même lorsque le hessien du lagrangien est indéfini. Plusieurs raisons plus pratiques expliquent ce choix : les sous-problèmes quadratiques deviennent strictement convexes et possèdent donc toujours au plus une solution ; la plupart des mises à jour quasi-Newton de la section 2.3.4 génèrent des matrices définies positives.

Le système linéaire (2.18) joue un rôle prépondérant au sein de l'algorithme 2.4.1. Pour être en mesure d'obtenir une solution de ce système, il faut que sa matrice de coefficients soit inversible. (Gould, 1985, Théorème 1.1) montre qu'il faut que $J(x_k)$ soit de plein rang ligne et que la matrice hessienne du lagrangien soit définie positive sur le noyau de $J(x_k)$ i.e.

$$d^T\nabla^2_{xx}\mathcal{L}(x_k;\lambda_k)d > 0, \quad \forall d \neq 0 \text{ tel que } J(x_k)d = 0. \tag{2.26}$$

Le théorème suivant que l'on peut retrouver, par exemple, dans (Nocedal et Wright, 2006, Théorème 18.4), énonce les conditions nécessaires à une convergence rapide de la méthode SQP.

Théorème 5. *Supposons (x_0, λ_0) suffisamment proches d'un minimum local (x^*, λ^*) satisfaisant la LICQ et les conditions* (2.26)*, alors les itérés (x_k, λ_k) générés par l'algorithme 2.4.1 convergent quadratiquement vers (x^*, λ^*).*

Lorsque la LICQ n'est pas satisfaite au point x_k, la recherche d'un pas $(\Delta x, -\Delta\lambda)$ peut s'avérer numériquement difficile. Pour remédier à ce problème, l'ajout d'un terme non nul dans le bloc $(2,2)$ de la matrice de coefficients est préconisé. On parle alors de méthode SQP régularisée (voir par exemple (Gill et Wong, 2011)). Ce terme prend généralement la forme d'un multiple de la matrice identité et transforme le système linéaire (2.18) en

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L}(x_k;\lambda_k) & J(x_k)^T \\ J(x_k) & -\delta_k I \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta\lambda \end{bmatrix} = - \begin{bmatrix} \nabla f(x_k) - J(x_k)^T\lambda_k \\ h(x_k) \end{bmatrix}, \tag{2.27}$$

où $\delta_k > 0$ est appelé paramètre de régularisation.

Récemment, il y a eu un intérêt considérable dans la formulation de méthodes SQP dites stabilisées. Celles-ci sont spécifiquement conçues pour améliorer le taux de convergence des méthodes SQP pour les problèmes dégénérés (Hager, 1999; Fernández et Solodov, 2010; Gill et Robinson, 2013). Les méthodes SQP stabilisées sont essentiellement locales, en ce sens que leur formulation et leur analyse de convergence mettent l'accent sur les propriétés de ces méthodes dans un voisinage de la solution. Pour les problèmes pour lesquels la LICQ et la condition du second ordre sont vérifiées à l'optimalité, une convergence locale superlinéaire peut être démontrée (Wright, 1998; Hager, 1999). Cependant, il n'y a aucune garantie de convergence vers une solution locale pour un point de départ arbitraire. Fernández et Solodov (2010) montrent que la méthode SQP stabilisée converge superlinéairement dans le cadre plus général d'une méthode de Newton de type stabilisée pour des problèmes variationnels en utilisant une condition de second ordre appropriée, qui se réduit à la condition du second ordre dans le cas de l'optimisation (Izmailov et Solodov, 2012).

KNITRO (Byrd *et al.*, 2006), SNOPT (Gill *et al.*, 2005) et FilterSQP (Fletcher et Leyffer, 1998) sont trois solveurs très connus implémentant des variantes des méthodes SQP. SNOPT suit une approche de recherche linéaire dans laquelle la solution du sous-problème quadratique sert de direction de recherche suivant laquelle une fonction de mérite basée sur un lagrangien augmenté est minimisée. Dans l'article 1 du Chapitre 4, nous utilisons SNOPT pour des fins de comparaison avec notre propre algorithme. Finalement, FilterSQP implémente une approche de région de confiance dans laquelle la convergence est assurée par un filtre (Fletcher, 1982).

2.4.3 Les méthodes de points intérieurs

Cette revue de littérature des méthodes d'optimisation avec contraintes ne serait pas complète si l'on ne mentionnait pas les méthodes de points intérieurs. Cependant, puisque nos travaux ne portent pas directement sur ce type de méthode, nous ne nous étendrons pas.

Les méthodes de points intérieurs ont initialement été proposées par Frisch en 1955. L'idée générale de ces méthodes consiste à approcher d'une manière itérative la solution des problèmes d'optimisation avec contraintes d'inégalité de la forme

$$\begin{array}{ll} \min\limits_{x\in\mathbb{R}^n} & f(x) \\ \text{s.c.} & c(x) \geq 0 \end{array} \tag{2.28}$$

à partir de l'intérieur strict de l'ensemble réalisable $\{x \mid c(x) > 0\}$. Ces méthodes sont basées sur l'utilisation de fonctions barrière.

Définition 4. *Une fonction barrière $b(x)$ pour le programme* (2.28) *est une fonction non négative et continue sur le domaine admissible qui tend vers l'infini lorsque x tend vers la frontière à partir de l'intérieur du domaine, c'est-à-dire : $b(x) \to +\infty$ si $c(x) \to 0^+$*

Les premières méthodes de points intérieurs consistaient à résoudre une succession de problèmes barrières (P_{μ_k}), tout en faisant tendre le paramètre barrière μ_k vers 0

$$\min_{x\in\mathbb{R}^n} \Phi^B_{\mu_k}(x) \tag{P_{μ_k}}$$

où Φ^B est la fonction barrière logarithmique définie par :

$$\Phi^B_\mu(x) = f(x) - \mu \sum_{i=1}^{p} \log c_i(x). \tag{2.29}$$

Cette méthode fut, pour un temps, délaissée par les optimiseurs en raison du mauvais conditionnement de la matrice hessienne de Φ^B_μ lorsque le paramètre barrière devient trop petit ainsi que par l'apparition de nouvelles méthodes plus efficaces pour la résolution du problème (2.28) telles que les méthodes de programmation quadratique successive et du lagrangien augmenté. Le succès de l'algorithme de Karmarkar (1984) dans la résolution des problèmes d'optimisation linéaire (Kojima *et al.*, 1989) a suscité un nouvel intérêt de la part des optimiseurs pour les méthodes de points intérieurs primales-duales.

L'idée des méthodes de points intérieurs primales-duales réside dans la perturbation de la condition de complémentarité des conditions de KKT (2.3) pour le problème (2.28) par un paramètre $\mu > 0$ tout en maintenant $c(x) > 0$ et les multiplicateurs de Lagrange associés aux

contraintes d'inégalité $\lambda > 0$. Cette perturbation définit le système suivant :

$$\begin{aligned} &\nabla f(x) - \lambda^T \nabla c(x) = 0 \\ &\lambda_i c_i(x) = \mu, \quad i = 1, ..., p \\ &c(x) > 0 \\ &\lambda > 0. \end{aligned} \tag{2.30}$$

Sous des conditions de régularité standards, le théorème de Wright (1992) montre l'existence et l'unicité d'une trajectoire différentiable paramétrée par μ, appelée chemin central.

Théorème 6. *Supposons que l'intérieur strict de l'ensemble admissible soit non vide et que* x^* *soit une solution de* (2.28) *pour laquelle les conditions de KKT sont satisfaites pour un multiplicateur optimal* λ^**. Supposons aussi que les gradients des contraintes actives sont linéairement indépendants et que les conditions de complémentarité stricte* (2.7) *et du second ordre sont satisfaites en* (x^*, λ^*)*. Considérons la suite des problèmes barrière* (P_{μ_k}) *où* $\{\mu_k\}$ *est une suite de valeurs positives convergeant de façon monotone vers 0 lorsque* $k \to +\infty$*. Alors*

- *il existe une sous-suite de minima de* $\Phi^B_{\mu_k}(x)$ *convergeant vers* x^**,*
- *pour* k *suffisamment grand,* $\nabla^2 \Phi^B_{\mu_k}(x)$ *est défini positif,*
- *il existe une unique fonction continûment différentiable* $x(\mu)$ *constituée des minima de* $\Phi^B_\mu(x)$ *dans un voisinage de* $\mu = 0$
- $\lim_{\mu \to 0} x(\mu) = x^*$.

Ainsi, le point limite du chemin central est une solution primale-duale optimale (x^*, λ^*) du problème initial (2.28).

Plusieurs logiciels implémentent ces méthodes de points intérieurs avec succès. Citons, par exemple, KNITRO (Byrd *et al.*, 2006), IPOPT (Wächter et Biegler, 2006) et LOQO (Vanderbei, 1999). Il faut noter que KNITRO et LOQO sont des logiciels commerciaux tandis qu'IPOPT est open-source et fait partie de l'initiative COIN-OR (`www.coin-or.org`).

2.5 Résolution des systèmes linéaires provenant de l'optimisation

Les méthodes d'optimisation numérique pour résoudre les problèmes avec contraintes requièrent la solution de systèmes linéaires (système augmenté, de point de selle ou encore de KKT) de la forme

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ g \end{bmatrix}, \tag{2.31}$$

où H est une matrice symétrique $n \times n$ et A est une matrice $m \times n$. Par exemple, H pourrait être la matrice hessienne d'une fonction objectif, d'une fonction de mérite ou même d'une fonction lagrangienne et la matrice A, la matrice jacobienne d'un ensemble de contraintes. De tels systèmes apparaissent à chaque itération des méthodes SQP présentées dans la section 2.4.2 avec de nouvelles matrices A et H et un nouveau membre de droite à chaque itération. La résolution d'un tel système linéaire doit donc être la plus efficace possible afin que la méthode d'optimisation soit rapide.

Dans la suite de cette section, nous présentons deux options pour résoudre ces systèmes.

La première option consiste à résoudre (2.31) en factorisant sa matrice des coefficients que nous appelons dans la suite K. Pour cette matrice symétrique, une factorisation de Cholesky ne peut pas être employée puisqu'elle est en général indéfinie (à moins que $m = 0$ et H soit définie positive ou négative). Cependant, Bunch et Parlett (1971); Bunch et Kaufman (1977) proposent d'utiliser une factorisation symétrique indéfinie décomposant K en LBL^T, où L est une matrice triangulaire inférieure unitaire et B est bloc-diagonale avec des blocs de taille 1×1 et 2×2.

Dans certaines méthodes de points intérieurs pour la programmation linéaire ou non linéaire ainsi que dans certaines méthodes des moindres carrés régularisés (Saunders, 1996), un système symétrique quasi-défini (SQD) de la forme

$$\begin{bmatrix} H & A^T \\ A & -N \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} b \\ g \end{bmatrix}, \tag{2.32}$$

doit être résolu, où H et N sont des matrices symétriques définies positives. Bien que dans la méthode des moindres carrés linéaires ou dans la méthode des points intérieurs appliquée à la programmation linéaire, H et N soient des matrices diagonales, ce n'est pas toujours le cas. Les matrices SQD sont soit définies (si $n = 0$ ou $m = 0$) soit indéfinies mais sont dans les deux cas fortement factorisables (Vanderbei, 1995), c'est-à-dire qu'elles possèdent une factorisation de type Cholesky LDL^T pour laquelle la matrice diagonale D n'est pas nécessairement définie positive : celle-ci possède la même inertie que la matrice SQD du système (2.32). Au lieu d'utiliser une factorisation pour les matrices symétriques indéfinies pour factoriser K, Gill *et al.* (1996) montrent que prendre en compte la structure particulière des matrices SQD lors de la factorisation engendre des facteurs généralement beaucoup plus creux en un temps de calcul plus faible, tout en conservant une méthode numériquement stable.

La deuxième option consiste à résoudre les systèmes linéaires (2.31) ou (2.32) de manière itérative sans devoir factoriser de matrices. Parmi ces méthodes itératives, les méthodes de Krylov sont les plus utilisées pour résoudre $Kx = e$. Elles reposent sur la génération d'une

base orthogonale pour les sous-espaces de Krylov générés par K et e. Cette base orthogonale est généralement obtenue par la processus de Lanczos symétrique (Lanczos, 1952). Nous référons le lecteur à Gutknecht (2007) pour une introduction aux sous-espaces de Krylov.

Dans le cas d'un système $Kx = e$ impliquant une matrice K symétrique définie positive, la méthode de Krylov la plus célèbre est sans aucun doute la méthode du gradient conjugué (Hestenes et Stiefel, 1952). MINRES (Paige et Saunders, 1975) est, quant à elle, la plus adéquate lorsque K est symétrique indéfinie. Fong et Saunders ont montré que même lorsque K est définie positive, MINRES est susceptible de s'arrêter plus rapidement que le gradient conjugué.

Il existe une étroite connexion entre les systèmes linéaires présentés ci-dessus et les problèmes aux moindres carrés linéaires. Par conséquent, nous les décrivons brièvement dans cette section. En effet, un problème aux moindres carrés linéaire en norme euclidienne identifie une solution de

$$\min_{y \in \mathbb{R}^m} \ \tfrac{1}{2}\|A^T y - b\|^2, \tag{2.33}$$

où $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^n$. Le problème peut être carré, sous-déterminé, ou sur-déterminé, i.e. $n = m$, $n < m$ ou $n > m$. Les conditions d'optimalité de (2.33) sont alors données par

$$\begin{bmatrix} I & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} r \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{2.34}$$

où $r = b - A^T y$ est le vecteur des résidus. Par élimination de r dans la première équation de (2.34), y doit satisfaire un ensemble d'équations

$$AA^T y = Ab \tag{2.35}$$

appelées équations normales.

LSMR (Fong et Saunders, 2011) et LSQR (Paige et Saunders, 1982a,b) sont deux méthodes permettant de résoudre ce genre de problème aux moindres carrés. LSMR est équivalent à MINRES appliqué aux équations normales (2.35), tandis que LSQR est une variante du gradient conjugué appliqué à ces mêmes équations normales.

Lorsque A n'est pas de plein rang ligne ou possède un mauvais conditionnement, LSMR et LSQR peuvent résoudre alternativement le problème aux moindres carrés régularisés

$$\min_{y \in \mathbb{R}^m} \ \tfrac{1}{2} \left\| \begin{bmatrix} A^T \\ \lambda I \end{bmatrix} y - \begin{bmatrix} b \\ 0 \end{bmatrix} \right\|_2^2, \tag{2.36}$$

pour un certain paramètre de régularisation λ et cela sans utiliser la matrice augmentée $\begin{bmatrix} A^T \\ \lambda I \end{bmatrix}$, ni des vecteurs de taille $m + n$.

Pour conclure, il faut ajouter qu'Arioli et Orban (2013) ont étendu LSMR et LSQR à la résolution des problèmes aux moindres carrés préconditionnés

$$\min_{y \in \mathbb{R}^m} \ \tfrac{1}{2} \|A^T y - b\|^2_{H^{-1}}, \tag{2.37}$$

où $H \in \mathbb{R}^{n \times n}$ est une matrice symétrique définie positive. Ses conditions de KKT sont alors

$$\begin{bmatrix} H & A^T \\ A & 0 \end{bmatrix} \begin{bmatrix} r \\ y \end{bmatrix} = \begin{bmatrix} b \\ 0 \end{bmatrix}, \tag{2.38}$$

où $r = b - A^T y$ est le vecteur des résidus.

Les conditions de KKT d'un problème aux moindres carrés linéaire régularisé et préconditionné conduiraient à un système équivalent à (2.32). Arioli et Orban (2013) montrent que LSQR et LSMR peuvent aussi être utilisés pour résoudre efficacement ce système dans les cas où il existe des méthodes performantes pour résoudre des systèmes impliquant les matrices H et N. Pour de plus amples détails à propos des méthodes de Krylov, nous référons le lecteur au livre de Saad (2003) et à Arioli et Orban (2013) pour l'utilisation des méthodes de Krylov pour la résolution de systèmes SQD.

CHAPITRE 3 ORGANISATION DE LA THÈSE

Cette thèse a pour objectif de décrire, analyser et implémenter deux méthodes d'optimisation non linéaire qui ne requièrent aucune factorisation. Les besoins pour de telles méthodes se font de plus en plus sentir au sein de l'industrie. En effet, la complexité et la taille des problèmes à résoudre ne cessant d'augmenter, les méthodes basées sur des factorisations atteignent leurs limites. Les récents progrès faits dans le développement de méthodes itératives adaptées aux systèmes linéaires symétriques quasi-définis et le peu de méthodes sans factorisation existant dans la littérature ont motivé les développements présentés dans cette thèse. Le corps de celle-ci consiste en une succession d'articles constituant notre propre contribution.

Le premier article, correspondant au chapitre 4, présente la mise en œuvre d'un algorithme de lagrangien augmenté sans factorisation pour les problèmes avec contraintes d'égalité et d'inégalité. La motivation principale de cet article provient d'un problème d'optimisation de structure d'une aile d'avion. Dans de nombreux problèmes d'optimisation structurelle et multidisciplinaire, les coûts de calcul sont dominés par le calcul des gradients de la fonction objectif et des contraintes. Si le problème contient à la fois un grand nombre de variables et un grand nombre de contraintes, le calcul de la matrice jacobienne devient exorbitant. La méthode généralement employée pour éviter cet inconvénient est l'agrégation de contraintes utilisée conjointement avec la méthode adjointe. Malheureusement, cette réduction se fait au détriment du bon conditionnement des sous-problèmes et de la qualité de la solution finale. L'implémentation d'une méthode de lagrangien augmenté nous semblait alors être la voie à emprunter car elle nécessite la solution approchée d'une séquence de sous-problèmes avec contraintes de bornes seulement. Cette méthode est implémentée grâce à NLP.py, un environnement de développement d'algorithmes d'optimisation que nous présentons dans le Chapitre 7. Cette méthode ne nécessite que l'utilisation de produits de la jacobienne avec des vecteurs plutôt que de former la matrice au complet. Ces produits matrice-vecteur peuvent être obtenus à une fraction du coût de formation de la matrice complète. La méthode développée permet de résoudre des problèmes d'optimisation de très grande taille, non convexes et possédant des contraintes d'égalité et d'inégalité. De plus, comparativement aux méthodes d'optimisation avec factorisation, elle s'avère compétitive. Nous montrons aussi que notre approche réussit à résoudre des problèmes d'optimisation de structure avec des milliers de variables et contraintes alors que les méthodes avec factorisation échouent.

Dans le chapitre 5, nous présentons une procédure systématique de régularisation pour les problèmes d'optimisation ne comportant que des contraintes d'égalité. Celle-ci peut être

interprétée comme une méthode de lagrangien augmenté proximal. Notre méthode utilise ce lagrangien augmenté proximal comme fonction de mérite afin de promouvoir la convergence globale de notre algorithme qui se fond asymptotiquement en une méthode de programmation quadratique séquentielle stabilisée possédant des propriétés de convergence locale rapide. Grâce à l'utilisation d'approximations BFGS à mémoire limitée du hessien du lagrangien, le système linéaire rencontré à chaque itération de notre méthode est symétrique et quasi-défini (Vanderbei, 1995), permettant ainsi d'être résolu de façon inexacte et donc d'obtenir une implémentation sans factorisation de notre algorithme. L'inexactitude de la résolution est contrôlée par un critère d'arrêt facile à mettre en œuvre dans une méthode de Krylov, assurant ainsi que la direction obtenue est bien une direction de descente pour la fonction de mérite choisie. Grâce au lien entre la structure de point de selle des systèmes linéaires et les problèmes aux moindres carrés, une méthode de Krylov plus efficace qu'une simple application de MINRES (Paige et Saunders, 1975), est proposée. Les convergences globale et locale de cet algorithme y sont étudiées, démontrant une convergence superlinéaire. La méthode proposée dans cet article est implémentée en Python à l'aide de NLP.py. Des tests numériques indiquent que cette méthode est efficace et robuste, en particulier pour la résolution de problèmes pour lesquels la LICQ n'est pas respectée à la solution ou lors de la minimisation. De plus, nos expérimentations numériques montrent que l'approche proposée se compare très favorablement à IPOPT.

Les chapitres 6 et 7 concernent la description d'un écosystème de développement d'algorithmes d'optimisation en Python, baptisé NLP.py. Cet environnement s'adresse aussi bien aux chercheurs en optimisation qu'aux élèves désireux de découvrir ou d'approfondir les multiples facettes de l'optimisation. Lors du développement d'un nouvel algorithme, les chercheurs en optimisation convoitent un environnement qui leur permettra de le mettre en œuvre rapidement et efficacement. NLP.py permet l'accès à un ensemble de blocs constituant les éléments les plus importants des méthodes d'optimisation continue. NLP.py fournit également des mécanismes qui facilitent la conception et le prototypage de nouvelles méthodes basées sur ces blocs. Parmi ceux-ci se trouvent un ensemble de méthodes directes et itératives pour la résolution de systèmes linéaires. En outre, plusieurs recherches linéaires y sont implémentées de même que des méthodes résolvant des sous-problèmes avec contraintes de borne ou de région de confiance (Conn *et al.*, 2000). Grâce à ces blocs, le chercheur est en mesure de se concentrer sur la logique des algorithmes plutôt que sur les subtilités techniques de son implémentation. L'un des principaux objectifs de conception de NLP.py est de permettre aux utilisateurs d'expérimenter facilement de nouvelles idées ou des variantes de méthodes connues en échangeant un détail d'implémentation pour un autre. L'évaluation de l'impact de ces changements peut se faire dans NLP.py via des profils de performance. De plus, NLP.py offre

des capacités avancées de modélisation de problèmes d'optimisation continue. NLP.py est écrit dans une combinaison de deux langages de programmation afin d'allier la facilité d'écriture à l'efficacité de l'implémentation. La conception de nouvelles méthodes d'optimisation ainsi que la modélisation sont effectuées en Python (un puissant langage de programmation haut niveau) tandis que les tâches demandant plus de ressources sont implémentées en Cython, un surensemble de bas niveau de Python (Behnel *et al.*, 2011).

Finalement, nous résumons les contributions de cette thèse et discutons des perspectives de ce travail au chapitre 8 et une conclusion est présentée au chapitre 9.

CHAPITRE 4 ARTICLE 1 : A MATRIX-FREE AUGMENTED LAGRANGIAN ALGORITHM WITH APPLICATION TO LARGE-SCALE STRUCTURAL DESIGN OPTIMIZATION

Sylvain Arreckx
École Polytechnique de Montréal & GERAD, Canada
Andrew Lambe
University of Toronto Institute for Aerospace Studies, Toronto, ON, Canada.
Joaquim R. R. A. Martins
Department of Aerospace Engineering, University of Michigan, Ann Arbor, MI, USA
Dominique Orban
École Polytechnique de Montréal & GERAD, Canada

Abstract

In many large engineering design problems, it is not computationally feasible or realistic to store Jacobians or Hessians explicitly. Matrix-free implementations of standard optimization methods—implementations that do not explicitly form Jacobians and Hessians, and possibly use quasi-Newton approximations—circumvent those restrictions, but such implementations are virtually non-existent. We develop a matrix-free augmented-Lagrangian algorithm for nonconvex problems with both equality and inequality constraints. Our implementation is developed in the Python language, is available as an open-source package, and allows for approximating Hessian and Jacobian information. We show that our approach solves problems from the CUTEr and COPS test sets in a comparable number of iterations to state-of-the-art solvers. We report numerical results on a structural design problem that is typical in aircraft wing design optimization. The matrix-free approach makes solving problems with thousands of design variables and constraints tractable, even when function and gradient evaluations are costly.

4.1 Introduction

Aerospace engineered systems have been a prime target for the application of numerical optimization due to the large impact that weight reduction has on system performance. This

Ce chapitre correspond à l'article publié : S. Arreckx, A. Lambe, J. R. R. A. Martins and D. Orban (2015). A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization. *Optimization and Engineering*, 1–26.

is evident in the fuel mass required to launch a satellite into orbit and in the operating cost of modern transport aircraft, where the primary cost driver is the fuel.

One of the first such applications by aerospace engineers was structural design optimization, first proposed by Schmit (1960). The field was made possible by the advent of the finite-element method for structural analysis (Argyris, 1954; Turner et al., 1956), which enabled engineers to analyze much more complex geometries than was possible with analytic methods.

This work is motivated by aircraft wing design optimization using coupled, high-fidelity physics-based models of aerodynamics and structures (Kenway et al., 2014; Kenway and Martins, 2014). In such problems, the objective, constraints, and derivative evaluations are expensive because of the expense of the aerodynamic and structural analyses.

The design optimization problem of interest can be stated in the general form

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to } \; c(x) = 0, \; \ell \leq x \leq u, \tag{NLP}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable. For the time being, it is sufficient to note that any nonlinear program may be reformulated as (NLP).

Kennedy and Martins (2015) and Kenway and Martins (2014) solve aircraft design problems based on (NLP) using general-purpose Sequential Quadratic Programming (SQP) software, such as SNOPT (Gill et al., 2002). This approach is particularly effective when used in conjunction with the adjoint method, which computes first derivatives efficiently (Kenway and Martins, 2014; Lyu and Martins, 2014; Lyu et al., 2015).

Structural design optimization problems often include both a large number of constraints (e.g., a failure criterion for each structural element), and a large number of variables (e.g., the thickness of each structural element). In addition, the constraint Jacobian is typically dense because the structures being optimized are statically indeterminate—the static equilibrium equations alone are not sufficient to compute the stress in each element of the structure. As a result, the stress in a given element depends not only on the properties of that element, but also on how the load is transmitted throughout the structure. In consequence, each failure constraint depends on many design variables.

To make a factorization-based SQP approach feasible, Poon and Martins (2007) aggregate constraints. The technique is effective for solving problems with hundreds of structural failure constraints (Kenway and Martins, 2014; Kennedy and Martins, 2013) but causes the final structural mass to be overestimated because the objective is minimized on a subset of the feasible region (Poon and Martins, 2007).

There is a need for an optimization approach that does not require aggregation, yet is still computationally efficient in the presence of dense Jacobians, and a matrix-free approach is the natural choice. However, providing a matrix-free implementation of an SQP or interior-point method is not straightforward, and the current state of optimization software is insufficient. A matrix-free approach exploiting inexact Hessian-vector products was recently proposed by Hicken (2014) and subsequently applied to an aerodynamic shape optimization problem (Dener et al., 2015), but this approach is currently restricted to problems with equality constraints. The handling of inequality constraints and bounds presents a particular challenge.

A matrix-free SQP method would compute steps as inexact minimizers of constrained quadratic programs. If bound constraints are kept explicit as in SNOPT, each SQP subproblem is a quadratic program with both equality and inequality constraints, and it is not immediately apparent how to solve such problems efficiently using a matrix-free method, even if they are convex.

Iterative methods for equality-constrained quadratic subproblems, however, have been developed in recent years. Arioli and Orban (2013) propose families of iterative methods that are suitable for matrix-free SQP or interior-point methods. Building upon those methods, Arreckx and Orban (2015) describe a matrix-free implementation of a fully-regularized SQP-type method for equality-constrained problems related to that of Armand et al. (2012), and highlight its relationship with the standard augmented Lagrangian method. Previously, Gill and Robinson (2013) highlighted relationships between a primal-dual augmented Lagrangian and regularized SQP methods.

If the bounds are enforced by way of a logarithmic barrier, as in, for example, IPOPT (Wächter and Biegler, 2006) or KNITRO (Byrd et al., 2006), the subproblems are equality-constrained quadratic programs. IPOPT uses a line search filter scheme to guarantee global convergence while KNITRO uses a trust region with a merit function to ensure global convergence. A line-search variant of a matrix-free interior-point algorithm might employ an inexact Newton strategy on an appropriate formulation of the Newton equations. Numerous formulations are possible and can be regularized to mitigate ill-conditioning (Greif et al., 2014), but the linear systems must nevertheless be adequately preconditioned. Furthermore, the resulting steps must be checked to ensure continued progress toward optimality.

A matrix-free interior-point method of the trust-region type would suffer from the same ill-conditioning issue as the line-search variant. Furthermore, the step must be decomposed into components that lie in the null space and range space of the Jacobian. KNITRO uses a projected conjugate-gradient method (Gould et al., 2001) to compute the null-space component. Unfortunately, this approach requires accurate projections into the null space of the linear

equality constraints and this is best achieved if the Jacobian is explicitly available.

The augmented-Lagrangian method may be simpler to implement as it requires the approximate solution of a sequence of reasonably-conditioned bound-constrained subproblems. The subproblem solutions are used to update estimates of the Lagrange multipliers for the constraints of (NLP). Direction-finding subproblems involve solving linear systems with a coefficient matrix of the form $H = B + \rho J^T J$, where $\rho > 0$ is a penalty parameter. Efficient iterative methods, typically variants of the conjugate-gradient method, are available for this type of system. Indeed if B is positive definite on the nullspace of J, J has full row rank, and ρ is sufficiently large, H is symmetric and positive definite. Note that operator-vector products with H require operator-vector products with the constraint Jacobian and its adjoint, an operation that is often available in practical large-scale applications. The main disadvantage is that augmented-Lagrangian methods typically do not exhibit the favorable local convergence properties of SQP methods. However, the ease with which bound constraints and inequality constraints can be treated in the algorithm provides us with a convenient starting point for experimenting with matrix-free optimization.

Augmented Lagrangian methods are a staple of the optimization library of numerical methods. It would be impossible to give a complete list of references here. We refer the reader to the general textbooks of Bertsekas (1982), Conn et al. (2000) and Nocedal and Wright (2006) for a thorough literature review and a complete convergence analysis.

The algorithm proposed in this paper is released as part of the open-source package NLP.py (Orban, 2014), a programming environment for designing numerical optimization methods written in the Python programming language.

Few other implementations of the augmented Lagrangian method exist. Amongst them MINOS (Murtagh and Saunders, 1978, 2003), LANCELOT (Conn et al., 1992), and ALGENCAN (Andreani et al., 2008) are the most widely used. MINOS takes advantage of linear constraints in the problem and, like SNOPT, is a commercial product. Gawlik et al. (2012) develop a linearly-constrained augmented Lagrangian method for solving partial differential equation (PDE) constrained optimization problems as part as the Toolkit for Advanced Optimization (TAO) (Munson et al., 2012). Their matrix-free method is open-source, but only handles equality constraints. LANCELOT is designed to exploit the group-partially separable structure of the objective and constraints in order to gain efficiency when dealing with large sparse problems, which makes the code arduous to modify. Although the LANCELOT algorithm appears to require only matrix-vector products with the Jacobian, its current implementation requires the full Jacobian, and so technically does not qualify as matrix-free. ALGENCAN is similar to LANCELOT in that it uses a bound-constrained problem formulation, but

ALGENCAN handles inequalities in a different way. While LANCELOT replaces inequalities with equalities by way of slack variables, ALGENCAN keeps inequalities intact, and uses the Powell-Hestenes-Rockafellar (Rockafellar, 1973) augmented Lagrangian function, which leads to discontinuous second derivatives in the objective of the subproblems. Our work follows the approach of LANCELOT.

We now introduce the notation used in the remainder of this paper. The i-th component of the vector x is x_i, whereas x^k or $x^{k,j}$ stands for the vector x at outer iteration k or inner iteration (k, j). Define the Lagrangian

$$\mathcal{L}(x, \lambda) := f(x) + \lambda^T c(x), \tag{4.1}$$

where $\lambda \in \mathbb{R}^m$ is the current approximation to the vector of Lagrange multipliers associated to the equality constraints of (NLP). The augmented Lagrangian function is

$$\Phi(x; \lambda, \rho) := \mathcal{L}(x, \lambda) + \tfrac{1}{2}\rho\|c(x)\|_2^2. \tag{4.2}$$

We separate λ and ρ from x by a semicolon in the arguments of Φ to indicate that they are treated as parameters, and that Φ is really a function of the primal variables x. For future reference, note that

$$\nabla_{xx}\Phi(x; \lambda, \rho) = \nabla_{xx}\mathcal{L}(x, \lambda + \rho c(x)) + \rho J(x)^T J(x). \tag{4.3}$$

Finally, $P_\Omega(\bar{x})$ is the projection of the vector $\bar{x} \in \mathbb{R}^n$ into the set of simple bounds

$$\Omega := \{x \in \mathbb{R}^n \mid \ell \leq x \leq u\},$$

and is defined componentwise as $P_\Omega(\bar{x})_i = \text{median}(\ell_i, \bar{x}_i, u_i)$ for $i = 1, \ldots, n$.

The rest of this paper is organized as follows. Section 4.2 is devoted to a detailed description of our matrix-free algorithm and its implementation in the Python language. We provide numerical results on standard test problems in order to validate our implementation and to compare it to existing software. In Section 4.3 we explore in further detail the structural design optimization problem, and show the benefits of the matrix-free approach over SNOPT. Conclusions and future work are discussed in Section 4.4.

4.2 A Matrix-Free Augmented Lagrangian Implementation

4.2.1 Algorithmic Details

In this section, we briefly cover the algorithmic details of our augmented Lagrangian framework. Although the framework itself is standard and well known, the description allows us to highlight certain algorithmic choices and relate them to implementation specifics described in §4.2.2.

The k-th *outer iteration* of the augmented-Lagrangian algorithm consists in approximately solving the subproblem

$$\min_{x \in \mathbb{R}^n} \ \Phi(x; \lambda^k, \rho^k) \quad \text{subject to } \ell \leq x \leq u, \tag{4.4}$$

for fixed values of λ^k and ρ^k. We enforce satisfaction of the bound constraints explicitly in the subproblem. Each subproblem solution is followed by updates to λ^k, ρ^k, and subproblem stopping tolerances. Those updates are typically based on the improvement in constraint violation achieved in the most recent subproblem. Algorithm 4.2.1 summarizes this process, and follows Nocedal and Wright (2006, Algorithm 17.4) and Conn et al. (1992). The parameter updates in Step 4 are classic and follow updates implemented in LANCELOT (Conn et al., 1992) and ALGENCAN (Andreani et al., 2008).

At every outer iteration, (4.4) must be solved efficiently. In our implementation, two options are available. The first option follows LANCELOT and uses the method of Moré and Toraldo (1989). The iterate at the j-th inner iteration corresponding to the k-th outer iteration will be denoted $x^{k,j}$. We begin by building a quadratic model $q^{k,j}$ of Φ about $x^{k,j}$:

$$q^{k,j}(p) := \nabla_x \Phi(x^{k,j}; \lambda^k, \rho^k)^T p + \tfrac{1}{2} p^T B^{k,j} p,$$

where $B^{k,j}$ is a symmetric approximation of $\nabla_{xx} \Phi(x^{k,j}; \lambda^k, \rho^k)$ that need not be positive definite. Our implementation allows $B^{k,j}$ to be defined as a limited-memory BFGS or SR1 approximation (Nocedal and Wright, 2006). The step $p^{k,j}$ is then obtained as an approximate solution of the bound-constrained quadratic program

$$\min_{p \in \mathbb{R}^n} \ q^{k,j}(p) \quad \text{subject to } \ p \in \Omega_{k,j} \tag{4.9}$$

where $\Omega_{k,j} := \{p \in \mathbb{R}^n \mid x^{k,j} + p \in \Omega \text{ and } \|p\|_\infty \leq \Delta^j\}$ and $\Delta^j > 0$ is the current trust-region radius. Note that $\Omega_{k,j}$ is itself a box and there exist $\ell^{k,j}$ and $u^{k,j}$ such that $\Omega_{k,j} = \{x \in \mathbb{R}^n \mid \ell^{k,j} \leq x \leq u^{k,j}\}$. The step $p^{k,j}$ is accepted or rejected and the radius Δ^j is updated following standard trust-region criteria (Conn et al., 2000). Algorithm 4.2.2 summarizes the main steps

Algorithm 4.2.1 Outer Iteration—AUGLAG

1: Initialize $x^0 \in \Omega$, $\lambda^0 \in \mathbb{R}^m$, $\rho^0 > 0$, $\omega^0 > 0$ and $\eta^0 > 0$. Choose stopping tolerances $\epsilon_g > 0$ and $\epsilon_c > 0$. Set $k = 0$.

2: If the stopping conditions

$$\|x^k - P_\Omega(x^k - \nabla_x \mathcal{L}(x^k, \lambda^k))\|_\infty \leq \epsilon_g \quad \text{and} \quad \|c(x^k)\|_\infty \leq \epsilon_c$$

are satisfied, terminate with (x^k, λ^k) as final solution. Otherwise continue to step 3.

3: Compute x^{k+1} by approximately solving (4.4) and stopping as soon as

$$\|x^{k+1} - P_\Omega(x^{k+1} - \nabla_x \Phi(x^{k+1}; \lambda^k, \rho^k))\|_\infty \leq \omega^k.$$

4: If $\|c(x^{k+1})\|_\infty \leq \eta^k$, set

$$\lambda^{k+1} = \lambda^k + \rho^k c(x^{k+1}), \qquad \rho^{k+1} = \rho^k \tag{4.5}$$

$$\eta^{k+1} = \eta^k / (\rho^{k+1})^{0.9}, \qquad \omega^{k+1} = \omega^k / \rho^{k+1}. \tag{4.6}$$

Otherwise, set

$$\lambda^{k+1} = \lambda^k, \qquad \rho^{k+1} = 10\rho^k, \tag{4.7}$$

$$\eta^{k+1} = 0.1/(\rho^{k+1})^{0.1}, \qquad \omega^{k+1} = 1/\rho^{k+1}. \tag{4.8}$$

Increase k by one and return to step 2.

involved in the inner iteration.

Algorithm 4.2.2 Inner Iteration—SBMIN

1: Choose $\Delta^0 > 0$, $0 < \epsilon_1 \leq \epsilon_2 < 1$, and $0 < \gamma_1 < 1 < \gamma_2$. Set $j = 0$. Choose an initial guess $x^{k,0} \in \Omega$.

2: If

$$\|x^{k,j} - P_\Omega(x^{k,j} - \nabla\Phi(x^{k,j}; \lambda^k, \rho^k))\|_\infty \leq \omega^k,$$

terminate with $x^{k+1} := x^{k,j}$. Otherwise continue to Step 3.

3: Choose a symmetric $B^{k,j}$ and compute $p^{k,j}$ as an approximate solution of (4.9).

4: Compute $\Phi(x^{k,j} + p^{k,j}; \lambda^k, \rho^k)$ and define

$$r^j := \frac{\Phi(x^{k,j} + p^{k,j}; \lambda^k, \rho^k) - \Phi(x^{k,j}; \lambda^k, \rho^k)}{q^{k,j}(p^{k,j})}.$$

If $r^j \geq \epsilon_1$, then set $x^{k,j+1} = x^{k,j} + p^{k,j}$, otherwise set $x^{k,j+1} = x^{k,j}$.

5: Update the trust-region radius

$$\Delta^{j+1} = \begin{cases} \gamma_1\|p^{k,j}\|_\infty & \text{if } r^j < \epsilon_1 \\ \Delta^j & \text{if } r^j \in [\epsilon_1, \epsilon_2) \\ \max\{\Delta^j, \gamma_2\|p^{k,j}\|_\infty\} & \text{otherwise.} \end{cases}$$

Increment j by one and return to step 2.

In Algorithm 4.2.2, the initial guess $x^{k,0}$ may be simply set to the current outer iterate x^k or to a better approximation if one is available. In Step 3, $p^{k,j}$ is computed using a simple extension of the method of Moré and Toraldo (1991) to nonconvex quadratic programs. In contrast with the trust-region subproblem solver used in LANCELOT, the method of Moré and Toraldo (1991) allows the additions of many constraints at a time to the active-set estimate $\mathcal{A}(x) := \{i \mid x_i = \ell_i^{k,j} \text{ or } x_i = u_i^{k,j}\}$.

The face of $\Omega_{k,j}$ containing x is defined as

$$F_x := \left\{y \in \Omega_{k,j} \mid y_i = x_i \text{ if } x_i = \ell_i^{k,j} \text{ or } u_i^{k,j}\right\}.$$

The active-set method is divided into two stages. In the first stage, a projected gradient search is used to select a face of $\Omega_{k,j}$ that will act as a prediction of the optimal active set of (4.9). In the second stage, a reduced quadratic model $\hat{q}$ is formed involving only the free variables from the selected face, that is the components of p that are not at their bounds. This model may be written

$$\hat{q}(v) := q(p + Z_x v),$$

where Z_x is a prolongation operator consisting of columns of the identity that maps F_x to $\mathbb{R}^n$.

This reduced quadratic is then approximately minimized unconstrained using the conjugate gradient method to yield a search direction $d = Z_x v$. If a direction of negative curvature is detected during the conjugate gradient iterations, we follow this direction to the boundary of $\Omega_{k,j}$. A projected line search is then performed along d to ensure sufficient decrease, and satisfaction of the bound and trust-region constraints. Both the projected gradient search and the conjugate gradient algorithm are designed to terminate early and promote fast progress. We employ the same stopping conditions as Moré and Toraldo (1991).

The *binding set* at x is defined by

$$\mathcal{B}(x) := \{i \mid (x_i = \ell_i^{k,j} \text{ and } \partial_i q(x) \geq 0), \text{ or } (x_i = u_i^{k,j} \text{ and } \partial_i q(x) \leq 0)\}.$$

If the binding set at the iterate resulting from the projected search along the conjugate gradient direction coincides with the active set identified in the first stage, the conjugate gradient iterations are resumed to enforce further descent. Algorithm 4.2.3 summarizes the main steps involved in this active-set method. We refer the reader to (Moré and Toraldo, 1991) for more details on projected searches.

Algorithm 4.2.3 BQP

1: Set $t = 0$. Choose $\kappa > 0$, $\zeta > 0$ and $\tau \in (0, 1)$. Compute a feasible starting point p^0 for (4.9).

2: If
$$\|p^t - P_{\Omega_{k,j}}(p^t - \nabla q^{k,j}(p^t)))\|_\infty \leq \tau ||\nabla q^{k,j}(p^0)||_\infty,$$
terminate with $p^{k,j} \leftarrow p^t$. Otherwise continue to Step 3.

3: Generate projected gradient iterates w^m starting with $w^0 = p^t$ until either
$$\mathcal{A}(w^m) = \mathcal{A}(w^{m-1}) \quad \text{or} \quad q^{k,j}(w^{m-1}) - q^{k,j}(w^m) \leq \kappa \max_{1 \leq r < m} \{q^{k,j}(w^{r-1}) - q^{k,j}(w^r)\}.$$

4: Generate conjugate gradient iterates v^s to minimize $\hat{q}^{k,j}$ on F_{w^m} starting from $v^0 = 0$ until
$$\hat{q}^{k,j}(v^{s-1}) - \hat{q}^{k,j}(v^s) \leq \zeta \max_{1 \leq r < s} \{\hat{q}^{k,j}(v^{r-1}) - \hat{q}^{k,j}(v^r)\}$$
or negative curvature is detected. Let $d = Z_{w^m}(v^s - v^0)$.

5: If d is a direction of negative curvature, find the smallest $\gamma > 0$ such that $p^t + \gamma d$ is at the boundary of $\Omega_{k,j}$. Otherwise perform a projected line search to find a step length γ that satisfies an Armijo condition.

6: Set $p^{t+1} = P_{\Omega_{k,j}}(p^t + \gamma d)$.

7: If $\mathcal{B}(p^{t+1}) = \mathcal{A}(p^{t+1})$, tighten ζ and go back to Step 4, restarting the iterations from v^s.

8: Increment t by one and return to step 2.

In practice, several improvements related to the management of the trust region can increase

the efficiency of Algorithm 4.2.2. Two such improvements turned out to be effective in our implementation. The first is a non-monotone descent strategy (Toint, 1997). As described, Algorithm 4.2.2 enforces a monotone descent in $\Phi(\cdot; \lambda^k, \rho^k)$. In a non-monotone trust-region algorithm, a trial point may be accepted even if it results in an increase in Φ. However, a sufficient decrease is required after a prescribed number of iterations, which is 10 in our implementation.

The second improvement is the simplified version of the backtracking strategy of Nocedal and Yuan (1998) described by Conn et al. (2000). If $p^{k,j}$ is rejected at Step 4 of Algorithm 4.2.2, we perform an Armijo line search along $p^{k,j}$ instead of recomputing a new trust-region step. We impose a maximum of five backtracking iterations. If the line search is unsuccessful, $x^{k,j}$ remains the current iterate, the trust-region radius is reduced, and a new trust-region step is computed.

The second option to solve (4.4) is to use an existing method for bound-constrained problems, and our method of choice for this task is TRON (Lin and Moré, 1998). TRON is an active-set method similar in spirit to the method of Moré and Toraldo (1991) that iteratively determines a current working set by way of a projected gradient method, and explores faces of the feasible set using a Newton trust-region method. In its default implementation, TRON has the significant disadvantage that it requires the explicit Hessian in order to compute an incomplete Cholesky preconditioner to speed up the conjugate gradient iterations. We modified TRON so that only Hessian-vector products are required. This modification also allows us to use quasi-Newton approximations in place of the true Hessian. With this modification, the incomplete Cholesky factorization is made impossible, since we have no access to the Hessian. Matrix-free preconditioners, such as those of De Simone and di Serafino (2014) could be applied, but our current implementation uses no preconditioner in the conjugate gradient iterations.

4.2.2 Implementation

We implement the AUGLAG solver (Algorithms 4.2.1–4.2.3) in the Python language as part of the NLP.py development environment for linear and nonlinear optimization (Orban, 2014). Optimization problems are only accessed to evaluate the objective and its gradient, and to compute operator-vector products with the Hessian of $\mathcal{L}(x, \lambda)$ and the constraint Jacobian. NLP.py is open source and available at `https://github.com/dpo/nlpy`.

First derivatives must be provided. Second derivatives may be provided if they are available. However, in some applications, such as that described in §4.3, the Hessian of the augmented Lagrangian cannot be computed even in the form of Hessian-vector products, and we must be

content with quasi-Newton approximations. Following the notation of Martínez (1988), the Broyden class of secant updates can be written as

$$S^{k,j+1} = S^{k,j} + \Delta_2(s, y, S^{k,j}, v), \tag{4.10}$$

where $S^{k,j}$ and $S^{k,j+1}$ are the current and updated approximations, respectively,

$$\Delta_2(s, y, S, v) = \frac{(y - Ss)v^T + v(y - Ss)^T}{v^T s} - \frac{(y - Ss)^T s}{(v^T s)^2} vv^T, \tag{4.11}$$

for some choice of $v \in \mathbb{R}^n$, called the *scale* of the update, and $s := x^{k,j+1} - x^{k,j}$. The vector y is chosen so that the update $S^{k,j+1}$ satisfies a secant equation $S^{k,j+1}s = y$. In the BFGS and SR1 updates, v is defined by $v = y + (y^T s/s^T Ss)^{\frac{1}{2}} Ss$ and $v = y - Ss$, respectively.

For conciseness, in the following, we denote $\nabla f_{k,j} := \nabla f(x^{k,j})$, $J_{k,j} := J(x^{k,j})$, and $c_{k,j} := c(x^{k,j})$. If $\varphi : \mathbb{R}^n \to \mathbb{R}^n$ is a smooth function such that $S^{k,j+1}$ should approximate $\nabla\varphi(x^{k,j+1})$, then the choice $y := \varphi(x^{k,j+1}) - \varphi(x^{k,j})$ is appropriate. The first possibility is to ask $S^{k,j+1}$ to approximate $\nabla_{xx}\Phi(x^{k,j+1}; \lambda^k, \rho^k)$, and in that case, we should select

$$\begin{aligned} y &:= \nabla_x \Phi(x^{k,j+1}; \lambda^k, \rho^k) - \nabla_x \Phi(x^{k,j}; \lambda^k, \rho^k) \\ &= \nabla f_{k,j+1} - \nabla f_{k,j} + J_{k,j+1}^T(\lambda^k + \rho^k c_{k,j+1}) - J_{k,j}^T(\lambda^k + \rho^k c_{k,j}). \end{aligned}$$

However, approximating (4.3) as a monolithic Hessian without exploiting its structure leads to poor numerical behavior. Because we assume that exact first derivatives are available, products with $J(x)$ and $J(x)^T$ may be evaluated, and it remains to approximate the Hessian of (4.1), as suggested by Dennis Jr. and Walker (1981) in the context of nonlinear least-squares problems using the DFP secant method. Martínez (1988) generalizes this DFP Hessian approximation to the Broyden class of secant methods, in particular to BFGS and SR1. In view of (4.3), the structured quasi-Newton update takes the form

$$B^{k,j+1} \approx \nabla_{xx}\Phi(x^{k,j+1}; \lambda^k, \rho^k) = \nabla_{xx}\mathcal{L}(x^{k,j+1}, \lambda^k + \rho^k c_{k,j+1}) + \rho^k J_{k,j+1}^T J_{k,j+1}.$$

We therefore set $B^{k,j+1} := S^{k,j+1} + \rho^k J_{k,j+1}^T J_{k,j+1}$ and we seek an update

$$S^{k,j+1} \approx \nabla_{xx}\mathcal{L}(x^{k,j+1}, \lambda_{k,j+1}^\sharp)$$

that satisfies a secant equation, where $\lambda_{k,j+1}^\sharp := \lambda^k + \rho^k c_{k,j+1}$. The relevant function φ is now

$\varphi(x) := \nabla_x \mathcal{L}(x, \lambda^\sharp_{k,j+1})$, and the appropriate secant equation is

$$\begin{aligned} S^{k,j+1} s &= \nabla_x \mathcal{L}(x^{k,j+1}, \lambda^\sharp_{k,j+1}) - \nabla_x \mathcal{L}(x^{k,j}, \lambda^\sharp_{k,j+1}) \\ &= \nabla f_{k,j+1} - \nabla f_{k,j} + (J_{k,j+1} - J_{k,j})^T (\lambda^k + \rho^k c_{k,j+1}). \end{aligned} \tag{4.12}$$

The updated $S^{k,j+1}$ is then defined as in (4.10).

In practice, AUGLAG accepts problems with a mixture of general equality and inequality constraints and transforms the latter into non-negativity constraints, i.e., $c_{\mathcal{E}}(x) = 0$ and $c_{\mathcal{I}}(x) \geq 0$. We subsequently add slack variables to obtain constraints of the form

$$c_{\mathcal{E}}(x) = 0, \quad c_{\mathcal{I}}(x) - t = 0, \quad t \geq 0, \quad \ell \leq x \leq u.$$

The augmented Lagrangian (4.2) becomes

$$\Phi(x, t; \lambda, \rho) := f(x) + \lambda^T \begin{bmatrix} c_{\mathcal{E}}(x) \\ c_{\mathcal{I}}(x) - t \end{bmatrix} + \tfrac{1}{2}\rho \left\| \begin{bmatrix} c_{\mathcal{E}}(x) \\ c_{\mathcal{I}}(x) - t \end{bmatrix} \right\|_2^2.$$

The latter augmented Lagrangian is iteratively minimized subject to the bounds $t \geq 0$, $\ell \leq x \leq u$.

In the presence of inequalities, $\Phi(x, \cdot; \lambda, \rho)$ is a convex quadratic function of t. Every time Algorithm 4.2.2 identifies a new inner iterate $(x^{k,j}, t^{k,j})$, we may further minimize Φ in t subject to $t \geq 0$. This yields the *magical step* (Conn et al., 1999, 2000)

$$t_i := \max\left(0, \frac{\lambda_i}{\rho} + c_i(x^{k,j})\right), \quad i \in \mathcal{I}.$$

Finally, our solver may perform an automatic scaling of the problem. This procedure closely follows the one provided in IPOPT (Wächter and Biegler, 2006), which is a scalar rescaling of the objective and constraint functions that ensures that the infinity norm of the gradient at the starting point after projection onto the bounds is less or equal to a given threshold value (100 in our implementation).

4.2.3 Benchmarks

The numerical results were obtained on a 2.4 GHz MacBook Pro with 4 GB of memory running Mac OS X 10.7. We report our results using the performance profiles of Dolan and Moré (2002).

We first present a comparison of our inner solver, SBMIN, versus the bound-constrained optimization code TRON (Lin and Moré, 1998) on all the bound-constrained problems from the COPS 3.0 collection (Dolan et al., 2004) and from the CUTEr collection (Gould et al., 2003). This results in 255 problems, all of which were used in their default dimension. Each problem is given a limit of 3000 iterations and 1 hour of CPU time. The automatic problem scaling procedure available in NLP.py is disabled for the AUGLAG and SBMIN results because none of the codes we compared to perform scaling of the problem.

By default TRON terminates the iterations as soon as

$$\|x^k - P_\Omega(x^k - \nabla f(x^k))\|_2 \leq 10^{-7} \|x^0 - P_\Omega(x^0 - \nabla f(x^0))\|_2.$$

In order to make a fair comparison between the two solvers, we adjusted TRON's stopping criterion such that SBMIN and TRON stop as soon as the relative *infinity* norm of the projected gradient is below 10^{-7}. For both algorithms, the initial trust region radius is set to

$$\Delta^0 = \tfrac{1}{10}\|x^0 - P_\Omega(x^0 - \nabla f(x^0))\|_\infty.$$

All other parameters for TRON are set to their default values. For SBMIN, we set $\epsilon_1 = 10^{-4}$, $\epsilon_2 = 0.9$, $\gamma_1 = 0.25$ and $\gamma_2 = 2.5$. In the bound-constrained quadratic program solver BQP, ζ is set to 10^{-3} and when tightened, to 10^{-5}, and $\kappa = 0.1$. These values are chosen because they result in good overall performance compared to other values we have explored.

When limited-memory quasi-Newton approximations of the Hessian are employed, all optimization codes are run with the same number of pairs in the history: 3 for LBFGS, and 5 for LSR1.

Figure 4.1 shows performance profiles in terms of number of iterations and of Hessian-vector products. The results indicate that TRON is slightly more robust than SBMIN, and requires substantially fewer iterations and Hessian-vector products to converge. In this regard, it appears that enforcing the bound constraints at the level of the nonlinear problem as in TRON, instead of at the quadratic trust-region subproblem level, as in SBMIN, pays off in terms of efficiency.

We now compare the two variants of our matrix-free augmented-Lagrangian implementation AUGLAG, one using SBMIN as inner solver (AUGLAG-SBMIN) and the other one using TRON (AUGLAG-TRON), to LANCELOT A (Conn et al., 1992).

Because of the matrix-free nature of our algorithm and in order to do fair comparisons, partial group separability is disabled in LANCELOT, we use a box trust region, and disable the

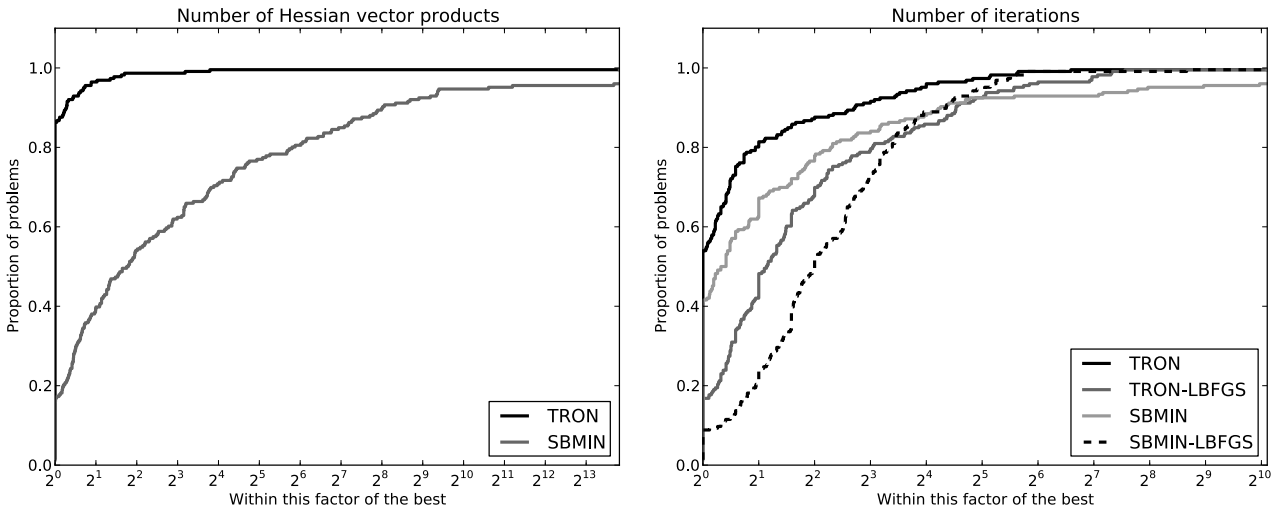

Figure 4.1 Comparison between SBMIN and TRON and their BFGS versions in terms of number of iterations and Hessian-vector products. Note that TRON-LBFGS and SBMIN-LBFGS don't appear in the left plot since they don't use any Hessian vector-products.

preconditioner in the conjugate gradient method. Furthermore, the Cauchy point calculation option was set to "approximate". All other options are set to their default values. Finally, for both solvers, the relative stopping tolerances, on the infinity norm of the projected gradient and constraint violation, are set to 10^{-7}. The initial trust region radius is set to

$$\Delta^0 = \tfrac{1}{10}\|(x,t)^0 - P_\Omega((x,t)^0 - \nabla\Phi((x,t)^0;\lambda^0,\rho^0))\|_\infty,$$

where λ^0 is a least-square estimate of the Lagrange multipliers.

Finally, we compare the algorithms on all problems from the COPS 3.0 collection (Dolan et al., 2004) and from the CUTEr collection (Gould et al., 2003) which possess at least one equality constraint or at least one bound constraint. This amounts to 675 problems. Again, a CPU time limit of 1 hour and an iteration count limit of 3000 is imposed. Figure 4.2 summarizes the performance of LANCELOT A and AUGLAG. The figure only reports the number of iterations because the LANCELOT A interface doesn't provide the number of Hessian-vector products required. The results indicate that AUGLAG-TRON is more robust than the two other codes when using either exact Hessian or quasi-Newton approximations. Both versions of AUGLAG perform slightly better than LANCELOT A when using exact derivatives. With LSR1 update, LANCELOT A, AUGLAG-SBMIN, and AUGLAG-TRON perform well.

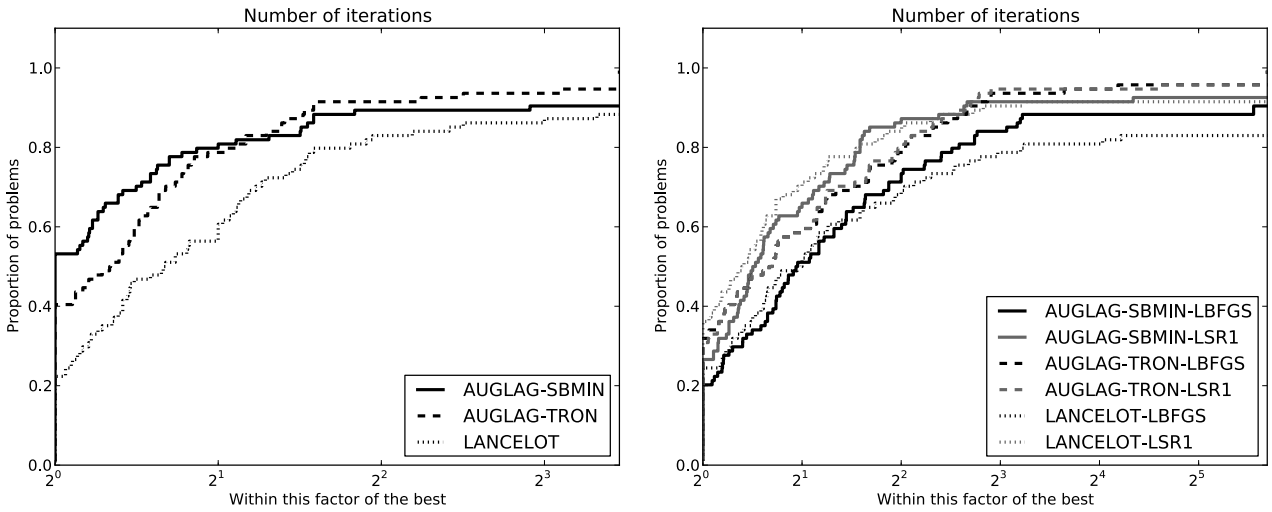

Figure 4.2 Comparison between AUGLAG-SBMIN, AUGLAG-TRON and LANCELOT A in terms of number of iterations with exact second derivatives (left) and with quasi-Newton approximations (right).

4.3 Structural Design Optimization Application

We now turn to a particular area of application for our matrix-free algorithm: aircraft structural design. Reducing the structural weight improves the fuel efficiency of the aircraft and therefore influences both the operating cost to the airline and the environmental impact of air transportation. Our goal is to minimize the mass of the structure subject to failure constraints. While many structural optimization problems are formulated with compliance (strain energy) constraints, the resulting solutions often show stress concentrations that would result in failure if the real structure were designed in that way. Therefore, optimization subject to failure constraints is more practical from an engineering design perspective. We start by describing the optimization problem formulation and how a matrix-free optimizer is helpful in this case before discussing the structural design optimization results.

4.3.1 Problem Formulation and Derivative Evaluations

Structural analysis involves the solution of static equilibrium equations in the form of a discretized PDE so this problem may be interpreted as a special case of PDE-constrained optimization. However, the stress constraints place further restrictions on the optimal set of state variables, and eliminating the discretized PDEs does not eliminate all of the constraints involving state variables. The full-space (Biros and Ghattas, 2005) or *simultaneous analysis and design* (SAND) problem (Haftka and Kamat, 1989; Martins and Lambe, 2013) is stated

as

$$\min_{x,y} \; F(x,y) \quad \text{subject to } C(x,y) \leq 0, \; R(x,y) = 0, \; \ell \leq x \leq u, \tag{SAND}$$

where $x \in \mathbb{R}^N$ are the design variables, $y \in \mathbb{R}^M$ are the state variables, $C : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^m$ are design constraints, and $R : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}^M$ are the discretized PDEs. Because we often use specialized software to solve the governing PDEs, and because N is usually much smaller than M, an alternative is to solve the reduced-space (Biros and Ghattas, 2005) or *nested analysis and design* (NAND) problem (Haftka and Kamat, 1989)

$$\min_{x} \; f(x) \quad \text{subject to } c(x) \leq 0, \; \ell \leq x \leq u, \tag{NAND}$$

where $y(x)$ is defined implicitly via $R(x, y(x)) = 0$, $f(x) := F(x, y(x))$, and $c(x) := C(x, y(x))$. Despite its smaller size, even (NAND) can have thousands of variables and constraints. Furthermore, the governing equations $R(x, y(x)) = 0$ must be re-solved for each new point computed by the optimizer, making function and gradient evaluation expensive. The chain rule and the implicit function theorem yield

$$\nabla c(x) = \nabla_x C(x, y(x)) + \nabla_x y(x) \nabla_y C(x, y(x)) \tag{4.13}$$

$$= \nabla_x C(x, y(x)) - \nabla_x R(x, y(x)) \nabla_y R(x, y(x))^{-1} \nabla_y C(x, y(x)) \tag{4.14}$$

where $\nabla_x C(x, y(x))$ denotes the transpose Jacobian of C with respect to x, i.e., the matrix whose columns are the gradients with respect to x of the component functions of C. We use a similar notation for the derivatives of R, and use "-1" for the inverse. Each matrix-vector product with $\nabla c(x)$ and $\nabla c(x)^T$ involves solving a linear system with coefficient matrix $\nabla_y R(x, y(x))$ and $\nabla_y R(x, y(x))^T$, respectively. Because both operations involve the solution of a large system of linear equations, the computational cost of a single matrix-vector product is similar to the cost of evaluating all the objective and constraint functions. Therefore, the success of the matrix-free approach for solving problem (NAND) hinges on keeping the sum of function evaluations and matrix-vector products small.

4.3.2 Approximating Jacobian Information

As mentioned in §4.2.2, exploiting the structure of the Hessian of the augmented Lagrangian leads to better performance on a wide range of problems. In particular, computing exact Jacobian-vector products within the trust-region solver and using a structured Hessian approximation to estimate the remaining terms is an effective strategy. However, this strategy can be too expensive when applied to structural design problems. Every time a Hessian-vector product is computed in the trust-region solver, two products with the Jacobian (one forward

and one transpose) are required. We have observed many instances in which the number of Jacobian-vector products needed to solve a given trust-region subproblem exceeds the number of constraints of the problem. Under these circumstances, if sufficient memory were available, it would be more efficient to form and store the entire Jacobian for computing these products than to compute the products from scratch. Therefore, we need to further refine the basic algorithm to reduce the number of expensive matrix-vector products.

We propose two different approaches for reducing the number of Jacobian-vector products in our matrix-free algorithm. Both approaches rely on using the Jacobian-vector products to create more accurate trust-region subproblem models using additional quasi-Newton matrix approximations. By using approximate Jacobian information in the trust-region subproblem, we prevent the number of Jacobian-vector products in any given iteration from becoming too large and keep the cost of solving the subproblem low. Note that exact Jacobian-vector products are still used to compute gradients of the Lagrangian and augmented Lagrangian function. Approximate Jacobian information is only used in the trust-region subproblem.

The first approach estimates the Hessian of the quadratic penalty term of the augmented Lagrangian function separately from the Hessian of the Lagrangian. We refer to this approach as the "split" quasi-Newton method. Briefly setting aside the structured quasi-Newton method of §4.2.2, we define $B_{\mathcal{L}} \approx \nabla^2_{xx}\mathcal{L}$ and $B_{\mathcal{I}} \approx \nabla^2_{xx}\frac{1}{2}\rho c(x)^T c(x)$. The gradient of the infeasibility function is simply $\rho J(x)^T c(x)$ so constructing the Hessian approximation is straightforward by splitting the gradient of the augmented Lagrangian function into the gradient of the Lagrangian and the gradient of the infeasibility function. We obtained the best results using the limited-memory SR1 approximation for $B_{\mathcal{L}}$ and the limited-memory BFGS for $B_{\mathcal{I}}$. The choice of a combination of quasi-Newton methods is informed by the fact that $\nabla^2_{xx}\frac{1}{2}\rho c(x)^T c(x) \approx J(x)^T J(x)$, a positive semidefinite matrix, near the optimal solution, while $\nabla^2_{xx}\mathcal{L}$ is not guaranteed to be definite near the optimal solution. To further improve the approximation provided by $B_{\mathcal{I}}$, we use a starting diagonal that is an approximation of the true diagonal of $J(x)^T J(x)$. The approximation is computed in the same way as the preconditioner proposed by De Simone and di Serafino (2014). Because both quasi-Newton approximations are limited-memory approximations, this approach is very memory-efficient for large optimization problems.

The second approach estimates the Jacobian matrix directly. In other words, we replace the true Jacobian-vector products for the algorithm outlined in §4.2.2 with the products of the same vectors with an approximate Jacobian matrix. In general, the Jacobian is not a square matrix, so alternative quasi-Newton approximations need to be used. Two such approximations are the two-sided rank-one (TR1) method, proposed by Griewank and Walther (2002), and

the adjoint Broyden method, proposed by Schlenkrich et al. (2010). Because the TR1 method requires more frequent updates to the Lagrange multipliers than we have available in our algorithm, we have selected the adjoint Broyden method for implementation. Unfortunately, no convergence theory exists for limited-memory quasi-Newton Jacobian estimates and it is not obvious how to initiate a robust limited-memory approximation. Therefore, we have chosen to implement a full-memory version of this approximation.

The basic adjoint Broyden update is given by the formula

$$A^{k,j+1} = A^{k,j} + \frac{\sigma^{k,j}\sigma^{k,j,T}}{\sigma^{k,j,T}\sigma^{k,j}} \left(J(x^{k,j+1}) - A^{k,j} \right) \tag{4.15}$$

where A is the approximate Jacobian and σ is an "adjoint search direction." Note that this update requires at least one (adjoint) Jacobian-vector product. Unlike traditional quasi-Newton methods, the choice of the search direction is not obvious. Schlenkrich et al. (2010) suggest several alternatives, from which we choose option (A), given by

$$\sigma^{k,j} = (J(x^{k,j+1}) - A^{k,j})s^{k,j} \tag{4.16}$$

where $s^{k,j} = x^{k,j+1} - x^{k,j}$, as the method to use with our algorithm. This particular choice of σ yields an update that is similar to the original TR1 update. Compared to the split quasi-Newton strategy, this strategy requires an additional Jacobian-vector product to compute $\sigma^{k,j}$. Despite the increase in required memory and higher cost of the update, this method has a distinct advantage over the split quasi-Newton approach in that the sparsity structure of any slack variables in the Hessian is preserved. That is, the block of $\nabla^2_{xx}\frac{1}{2}\rho c(x)^T c(x)$ associated with the slack variables is known exactly (an identity matrix) so it may be treated exactly in the Hessian-vector product. This approach leads to a much more accurate Hessian approximation than the split quasi-Newton method if the problem contains many slack variables.

We close this section with a few implementation details of the adjoint Broyden method. Similar to other quasi-Newton schemes, we reject the update if the denominator of the update term in (4.15) is sufficiently small, i.e., if $\sigma^T\sigma \leq 10^{-20}$. Our initial approximation $A^{0,0}$ is set to be the exact Jacobian $J_{0,0}$. While this strategy has a very high up-front cost, we found that it paid off on our test problem in terms of many fewer major iterations required by the optimization. We recognize that our strategy may not be sound for all problems, especially those in which the constraints are highly nonlinear. However, we expect the approach to be successful on many problems given the established robustness of quasi-Newton methods.

4.3.3 Optimization Results

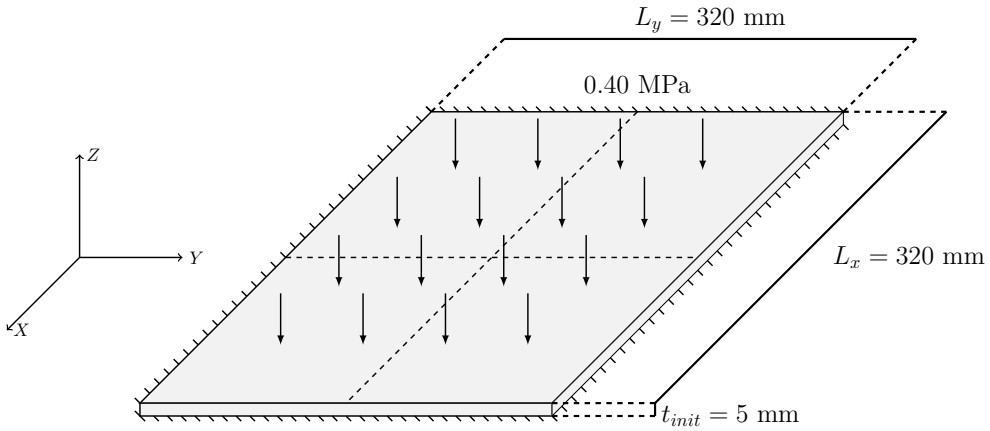

Figure 4.3 Geometry and load condition of plate mass minimization problem

We use the following test problem to compare our matrix-free algorithm against an optimizer that requires the full Jacobian. The problem is to minimize the mass of a square, metallic plate that is clamped on all sides and subject to a uniform pressure load, as shown in Figure 4.3. The structural analysis of the plate is performed using the finite-element program TACS (Kennedy and Martins, 2014) with third-order shell elements. The optimization problem is constrained so that the maximum von Mises stress on any of the plate elements does not exceed the material yield stress. The design variables of the problem are the thicknesses of each plate element. Minimum and maximum thicknesses are imposed on each element. To simplify the problem, we analyze only one quarter of the plate and apply symmetry boundary conditions on the unclamped edges. Since each structural element is associated with one design variable (its thickness) and one constraint (the stress), the number of structural elements, design variables, and constraints is the same for a given problem. Except for the design variable bounds, all constraints are nonlinear.

While this test problem does not represent a complete aircraft structure, it shares two challenging features of such structures. First, the structure is a shell structure subject to a distributed load. This type of structure requires higher-order two- or three-dimensional finite elements to be used for accurate analysis of the structural behavior. The resulting analysis is therefore much more expensive than analyses using one-dimensional elements due to the larger number of degrees of freedom. Second, and more importantly, the structure is not statically determinate and has many degrees of indeterminacy. This means that the full finite-element analysis must be completed in order to compute stresses and strains; no shortcuts can be taken in evaluating the failure constraints. In practice, this finite-element

analysis can be ill-conditioned so the NAND problem formulation (NAND) is used to hide the ill-conditioning from the optimizer.

Our benchmark optimizer for this test is the general-purpose optimizer SNOPT (Gill et al., 2002) which is accessed in Python through the pyOpt interface (Perez et al., 2012). SNOPT is an active-set SQP optimizer capable of solving nonlinear and nonconvex problems. While the full version of SNOPT has no limits on the number of variables or constraints in the problem, it is especially suited to problems with a large number of sparse constraints and few degrees of freedom. Like our optimizer, SNOPT does not require second derivatives because it approximates them using a limited-memory quasi-Newton method. Unlike our optimizer, SNOPT requires first derivative information from the objective and all constraint functions. Our optimizer just requires the gradient of the objective function and forward and transpose products with the constraint Jacobian.

Due to the design of the TACS software, we are able to accommodate both traditional optimizers like SNOPT and matrix-free optimizers. For our expression for the Jacobian of the reduced-space problem in (4.14), TACS provides modules for computing the action of $\nabla_x R(x, y(x))$ and $\nabla_x R(x, y(x))^T$ on vectors of appropriate length. The different partial derivatives of the constraints themselves are computed with respect to individual constraints, effectively providing column-wise evaluation of $\nabla_x C(x, y(x))$ and $\nabla_y C(x, y(x))$. The term $\nabla_y R(x, y(x))^{-1}$ is computed implicitly by a specialized, sparse, parallel, direct factorization method. Every time we multiply this inverse or its transpose by a vector, we solve the appropriate upper- and lower-triangular systems by substitution. When computing the full Jacobian for SNOPT, TACS exploits parallel structure in the adjoint method to compute multiple adjoint vectors at the same time. This feature is not needed by the matrix-free optimizer since only individual matrix-vector products are ever called for. However, this added awareness of parallel computing does tend to skew the run-time results in favour of SNOPT.

We use the following settings in our matrix-free optimizer. The LSR1 Hessian approximation with five pairs of vectors is used to estimate the Hessian of the Lagrangian. The adjoint Broyden approximation is used to estimate the constraint Jacobian, where the initial Jacobian is computed exactly. In the split quasi-Newton strategy, the LBFGS approximation with five pairs of vectors is used to estimate the feasibility Hessian. Both magical steps and Nocedal–Yuan backtracking are turned on in the nonlinear, bound-constrained solver. In SBMIN, a limit of 50 iterations is imposed to solve the quadratic model problem. (On this specific problem, we found that SBMIN was superior to TRON.) Finally, parallel computations are used in the adjoint Broyden approximation to allow the approximate Jacobian to be

stored in a distributed fashion.

For this optimization problem, we also introduced an update to the Lagrange multipliers, modified from the update specified by Algorithm 4.2.1, that we found to be effective at improving algorithm performance. The multiplier update now takes the form

$$\lambda^{k+1} = \lambda^k + \alpha^k \rho^k c(x^{k+1}) \tag{4.17}$$

where $0 \leq \alpha^k \leq 1$ is a chosen damping factor. Note that $\alpha^k = 1$ corresponds to the traditional update specified in Algorithm 4.2.1. In this damped update, α^k is computed as the solution to the convex minimization problem

$$\min_{\alpha^k} \ \frac{1}{2}||\nabla f(x^{k+1}) + J(x^{k+1})^T \lambda^{k+1}||_2^2 \quad \text{subject to } \ 0 \leq \alpha^k \leq 1. \tag{4.18}$$

The solution to Problem (4.18) is easily determined to be

$$\alpha^k = \text{median}\left(0, \frac{-\rho^k c(x^{k+1})^T J(x^{k+1})(\nabla f(x^{k+1}) + J(x^{k+1})^T \lambda^k)}{||\rho^k J(x^{k+1})^T c(x^{k+1})||_2^2}, 1\right). \tag{4.19}$$

If $J(x^{k+1})^T c(x^{k+1}) = 0$ then α^k is also set to zero. In practice, this modified update improves the multiplier estimates in the first few outer iterations. We also observe that α^k is chosen close to 1 after a few updates, suggesting that the traditional multiplier update is optimal when x and λ are near a solution.

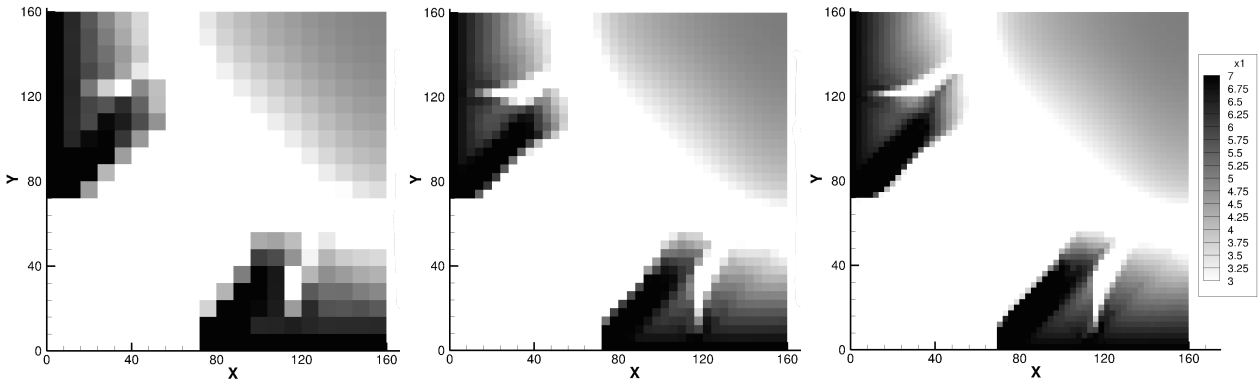

Figure 4.4 Final thickness distributions for the 400-, 1600-, and 3600-element plate problems. These solutions were all obtained by the matrix-free optimizer. The solutions from SNOPT for the 400- and 1600-element problems are nearly identical.

Example design solutions to the benchmark problem are shown in Figure 4.4 for three different mesh sizes, and the corresponding stress distributions are shown in Figure 4.5. In these figures,

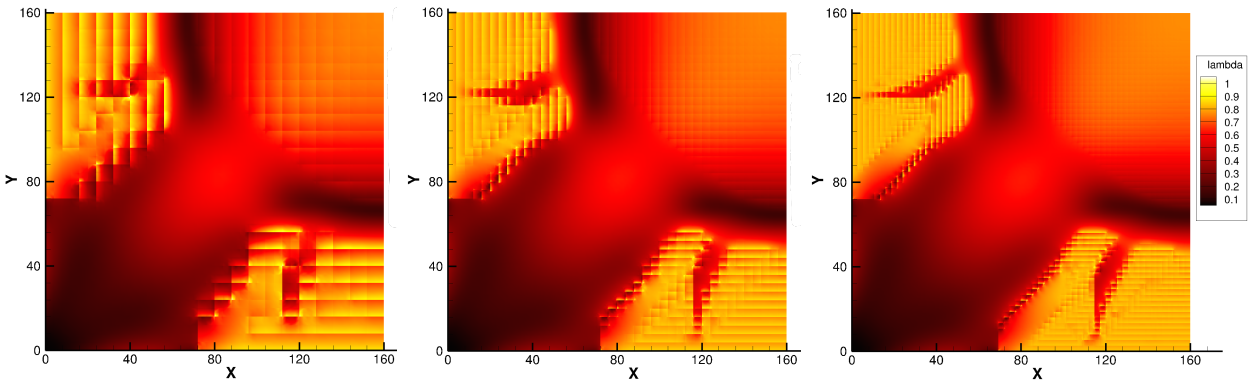

Figure 4.5 Stress distributions as a fraction of the local yield stress for the 400-, 1600-, and 3600-element plate problems.

the x- and y-axes of the plots correspond to the clamped edges of the plate. The built-up regions of the plate along the clamped edges and in the center of the plate are clearly visible. For every case in which both solvers found an optimal solution, both SNOPT and AUGLAG converged to similar final designs. The feasibility and optimality tolerances of both solvers were set to 10^{-5}, and both solvers achieved these tolerances at the final designs.

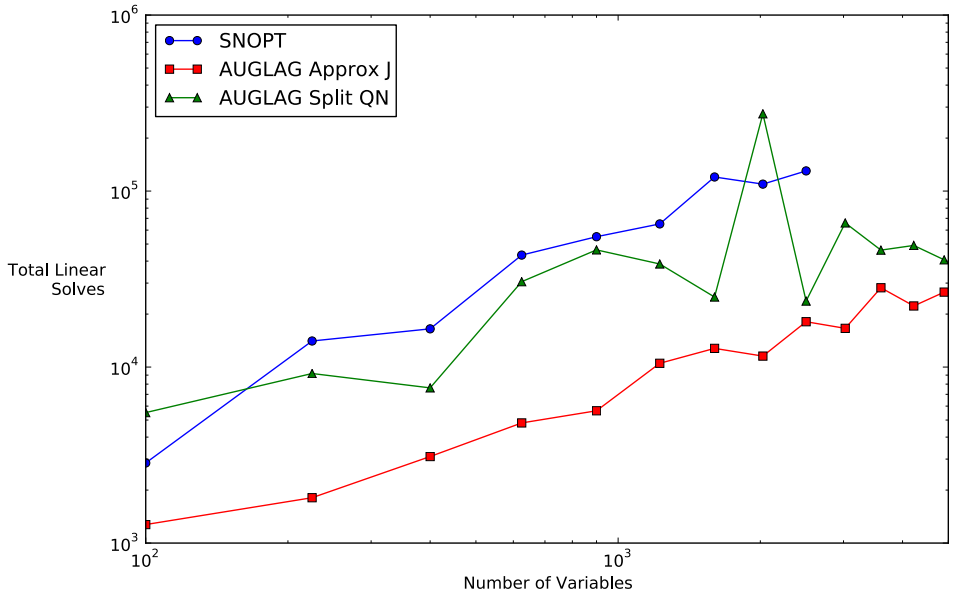

Figure 4.6 Number of finite-element linear solve operations required to solve the plate optimization problem

Figure 4.6 compares the number of finite-element linear systems—those involving $\nabla_y R(x, y(x))$—that are solved using each algorithm for a range of problem sizes. The finest mesh solved using either optimizer was 70×70 elements. The corresponding optimiza-

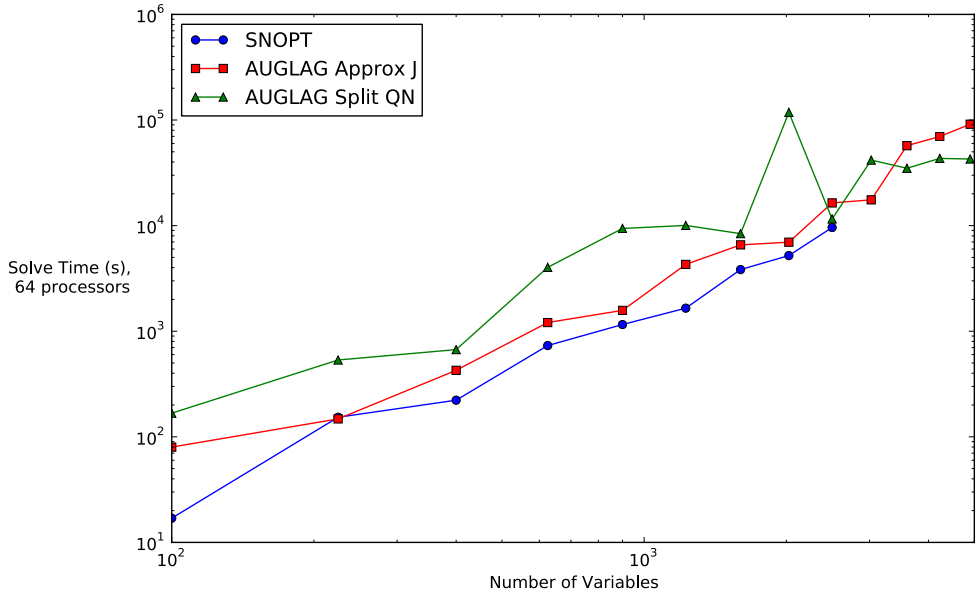

Figure 4.7 Run time to solve the plate optimization problem using 64 processors

tion problem had 4900 thickness variables and 4900 failure constraints. We use the number of finite-element linear system solutions as the primary metric for comparing the optimizers because solving the linear system associated with the finite-element method is the most costly operation in the optimization process. This operation occurs once to evaluate the failure constraints and once for every Jacobian-vector product. To form the entire Jacobian for SNOPT, a linear system is solved to obtain one column of the matrix so the matrix size determines the total work. Figure 4.6 demonstrates that, by not forming the Jacobian at each iteration, both matrix-free algorithms successfully reduce the number of expensive linear solve operations as the problem size increases. In fact, for problems with more than 1000 variables and constraints, the reduction produced by the approximate Jacobian approach is nearly one order of magnitude over SNOPT.

Figure 4.6 also shows that the matrix-free optimizer was able to solve larger optimization problems than SNOPT. SNOPT was unable to solve any problems for meshes larger than 50×50 elements due to a lack of memory. Each instance of the benchmark problem was solved in a distributed-memory computing environment. Because SNOPT was not designed to exploit this environment, it could only access the memory available to a single computing node, limiting the size of problem it could solve. We emphasize, however, that this is an artifact of the implementation of the SNOPT algorithm and not a fundamental limitation of the algorithm itself. There is no reason why an active-set SQP algorithm could not be developed to exploit the distributed-memory computing environment used to solve this problem.

Nevertheless, both matrix-free strategies lend themselves to more memory-efficient implementations. The reason for the high memory usage of SNOPT seems to be the symbolic factorization of the Jacobian as part of the active-set SQP algorithm. In the approximate Jacobian implementation of AUGLAG, we need to store the matrix, but we do not need to factorize it. While Message Passing Interface (MPI) standard commands are used, via the mpi4py library, to distribute the stored matrix across multiple nodes and compute matrix-vector products in parallel, a sequential implementation of the algorithm should be capable of solving the problem sizes shown here, though with a longer run time. In the split quasi-Newton implementation of AUGLAG, only limited-memory matrix approximations are used, and no special provisions are made for parallel computing. Therefore, if the optimizer were restricted to run on a single processor, we would expect the run time of that implementation to be identical.

Figure 4.7 shows a wall-time comparison for solving the optimization problems using 64 processors. Comparing Figures 4.6 and 4.7, the large reduction in linear system solve operations does not translate into reduced run time. In fact, SNOPT is still the fastest optimizer for the problem sizes that it is able to solve. We attribute this behavior to two causes. First, as mentioned above, the TACS solver is able to parallelize the (implicit) multiplication of $\nabla_y R(x, y(x))^{-1}$ by multiple right hand sides, reducing the time needed to form a large Jacobian. In other words, TACS is able to solve multiple adjoint systems simultaneously. This is a special feature of the TACS solver. Second, SNOPT requires many fewer iterations than our augmented Lagrangian solver to find the solution in each case. Fewer iterations means fewer points for which the partial derivative matrices must be recomputed. While this cost is small in comparison to the cost of a linear solve operation, the increase in the number of iterations outweighs the reduction in linear solves for this choice of algorithm.

One implementation decision that does not exert too much influence on the run time is the choice of implementation language of the optimizer. Figure 4.8 shows the fraction of the run time spent computing the next point in the optimizer for each case. When using SNOPT, only a small fraction of the run time is spent in the optimizer unless the problem is large. This increase in run time is probably due to the additional work needed to factorize the Jacobian in the active-set SQP algorithm. For the approximate Jacobian version of AUGLAG, the optimizer appears to take up the majority of the run time of the optimization process. However, nearly all of this time is spent forming matrix-vector products with the approximate Jacobian. Python makes use of both distributed-memory parallel processing and compiled-language libraries to complete this operation, so it is unlikely that moving to a compiled language implementation would result in a large reduction in run time. For the split quasi-Newton version of AUGLAG, the fraction of the run time spent in the optimizer

decreases with increasing problem size. Because so little time is spent within the optimizer itself using this approach, replacing the Python implementation of the algorithm with a compiled-language implementation would not result in large reductions in wall time.

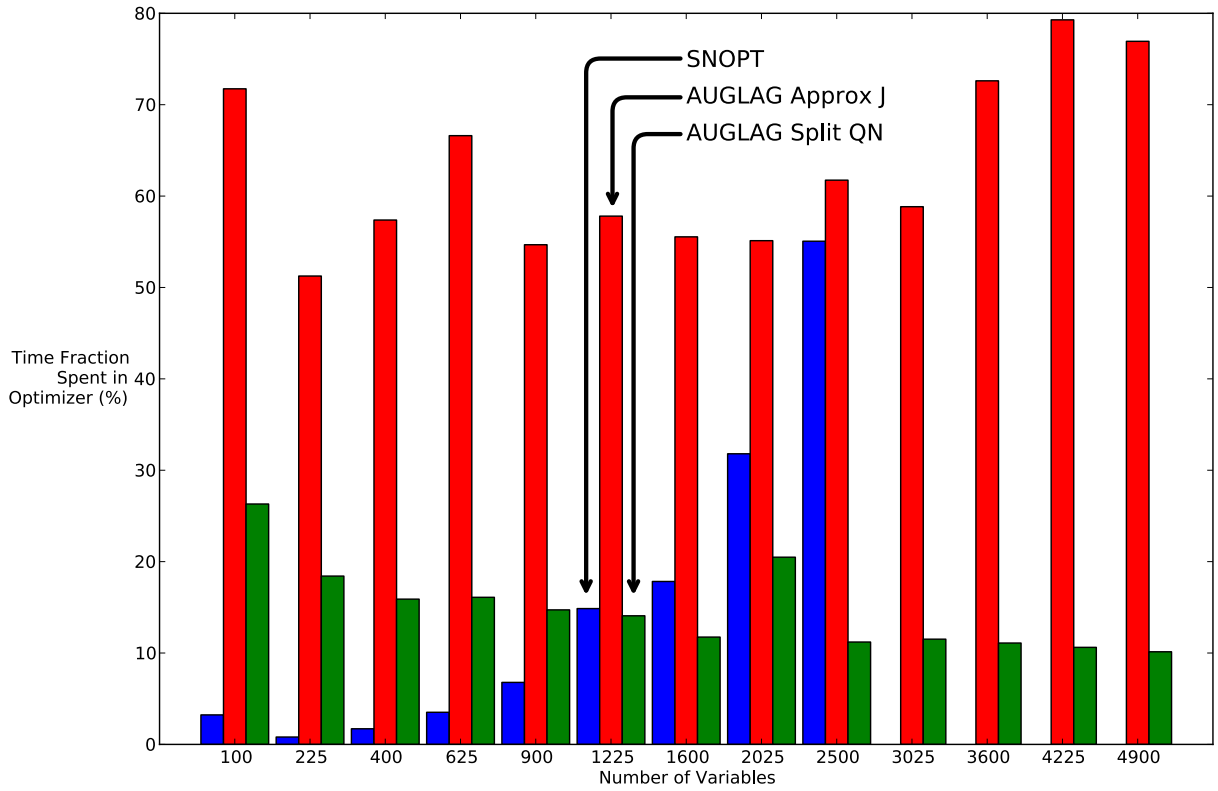

Figure 4.8 Percentage of wall time spent in optimizer for each instance of the plate problem.

These results effectively show the intrinsic trade-off of matrix-free optimization in engineering design applications. As demonstrated in Figure 4.6, if the engineering design problem has many constraints, using a matrix-free optimizer can lead to a massive reduction in the computational effort spent calculating gradient information. However, this reduction is offset by the overhead incurred by recomputing the design constraints and the relevant partial derivative matrices at more points in the design space. We suspect that changing the basic optimization algorithm from an augmented Lagrangian to an SQP or interior-point method would result in a matrix-free optimizer that is more competitive in terms of run time.

This example problem also raised the general issue of how to better exploit parallel computing within the optimization process. Because this particular problem is relatively small and dense, distributing the vectors used by the optimizer over multiple processors may not improve

algorithm performance at all. The cost of communicating results between processors would outweigh the performance benefits of parallelized linear algebra. (The main exception to this statement is the matrix-vector products with the full-memory approximate Jacobian.) Instead, parallel processing is most beneficial in performing the structural analysis and computing functions and their gradients, including matrix-vector products. The only parallel capability in the TACS code that was not exploited by our matrix-free optimizer was the ability to form a group of gradients, i.e., a Jacobian matrix, using parallel matrix multiplication. The equivalent operation in a matrix-free optimizer would be to compute several matrix-vector products at the same time for a given design point. The optimizer and quasi-Newton approximations would need to be carefully chosen and structured to allow for this setup. Because the main bottleneck in parallel processing is often communication between processors, identifying operations that require a high computational effort but little communication between processors is critical to exploiting the parallel processing environment.

4.4 Conclusion and future work

This paper details the implementation of a matrix-free optimizer based on the augmented Lagrangian algorithm. Benchmarking results indicate that this optimizer is competitive with LANCELOT on standard test sets. We then extend the algorithm to store approximate Jacobian information to reduce the required number of matrix-vector products. The extended algorithm is then applied to a test problem motivated by aircraft structural design. Our results indicate that the matrix-free optimizer successfully reduces the computational work of the structural analysis, represented by the number of linear system solutions, when the structural design problem has a large number of design variables and a large number of constraints. The reduction can be as much as an order of magnitude when the number of variables and the number of constraints are both large.

Our study also highlighted key areas for improvement in terms of the capability of matrix-free optimizers. Namely, providing a solver for quadratic problems with both equality and inequality constraints, or equality and bound constraints, is the key to developing a matrix-free SQP method. In addition, because the problems for which matrix-free optimizers are most useful rely heavily on parallel computing, the matrix-free optimizer itself should exhibit strong, scalable performance in a parallel computing environment.

In the near future, we hope to extend our engineering application to the design of aircraft wings, including coupled aerodynamic and structural optimization (Kenway et al., 2014; Kenway and Martins, 2014). The case of coupled aerodynamic and structural optimization is interesting because the features of the TACS solver that make it so fast on structural optimization

problems (specialized parallel matrix factorization and parallel solution of multiple adjoint linear systems) would be nullified in the multidisciplinary optimization problem. In that case, we expect the matrix-free optimizer to become a particularly attractive option.

Acknowledgements

We would like to thank Graeme J. Kennedy for his assistance in setting up the structural design problem shown in this work. The computations for that problem were performed on the General Purpose Cluster supercomputer at the SciNet HPC Consortium.

REFERENCES

R. Andreani, E. G. Birgin, J. M. Martínez, and M. L. Schuverdt (2008). On augmented Lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18(4):1286–1309. doi: 10.1137/060654797.

J. H. Argyris (1954). Energy theorems and structural analysis: A generalized discourse with applications on energy principles of structural analysis including the effects of temperature and non-linear stress-strain relations. *Aircraft Engineering and Aerospace Technology*, 26(10):347–356. doi: 10.1108/eb032482.

M. Arioli and D. Orban (2013). Iterative methods for symmetric quasi-definite linear systems—Part I: Theory. Cahier du GERAD G-2013-32, GERAD, Montréal, QC, Canada.

P. Armand, J. Benoist, and D. Orban (2012). From global to local convergence of interior methods for nonlinear optimization. *Optimization Methods and Software*, 28(5):1051–1080. doi: 10.1080/10556788.2012.668905.

S. Arreckx and D. Orban (2015). A regularized SQP method for degenerate equality-constrained optimization. Technical report, GERAD, Montréal, QC, Canada. In preparation.

D. P. Bertsekas (1982). *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press.

G. Biros and O. Ghattas (2005). Parallel Lagrange-Newton-Krylov-Schur Methods for PDE-Constrained Optimization. Part I: the Krylov-Schur Solver. *SIAM Journal on Scientific Computing*, 27(2):687–713. doi: 10.1137/S106482750241565X.

R. H. Byrd, J. Nocedal, and R. A. Waltz (2006). KNITRO: An integrated package for nonlinear optimization. In G. di Pillo and M. Roma, editors, *Large-Scale Nonlinear Optimization*, pages 35–59. Springer Verlag.

A. R. Conn, N. I. M. Gould, and P. L. Toint (1992). *Lancelot: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Publishing Company, Incorporated, 1st edition.

A. R. Conn, N. I. M. Gould, and P. L. Toint (2000). *Trust-region methods*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.

A. R. Conn, L. N. Vicente, and C. Visweswariah (1999). Two-step algorithms for nonlinear optimization with structured applications. *SIAM Journal on Optimization*, 9(4):924–947. doi: 10.1137/S1052623498334396.

V. De Simone and D. di Serafino (2014). a matrix-free approach to build band preconditioners for large-scale bound-constrained optimization. *Journal of Computational and Applied Mathematics*, 268(0):82–92. doi: 10.1016/j.cam.2014.02.035.

A. Dener, G. K. W. Kenway, Z. Lyu, J. E. Hicken, and J. R. R. A. Martins (2015). Comparison of inexact- and quasi-newton algorithms for aerodynamic shape optimization. In *53rd AIAA Aerospace Sciences Meeting*.

J. Dennis Jr. and H. Walker (1981). Convergence theorems for least-change secant update methods. *SIAM Journal on Numerical Analysis*, 18(6):949–987. doi: 10.1137/0718067.

E. D. Dolan and J. J. Moré (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213. doi: 10.1007/s101070100263.

E. D. Dolan, J. J. Moré, and T. S. Munson (2004). Benchmarking Optimization Software with COPS 3.0. Technical Report ANL/MCS-TM-273, Argonne national labaratory, Mathematics and Computer Science division.

E. Gawlik, T. Munson, J. Sarich, and S. M. Wild (2012). The TAO linearly constrained augmented Lagrangian method for PDE-constrained optimization. Preprint ANL/MCS-P2003-0112, Mathematics and Computer Science Division.

P. Gill and D. Robinson (2013). A globally convergent stabilized sqp method. *SIAM Journal on Optimization*, 23(4):1983–2010. doi: 10.1137/120882913.

P. E. Gill, W. Murray, and M. A. Saunders (2002). SNOPT: An SQP Algorithm for Large-Scale Constrained Optimization. *SIAM Journal on Optimization*, 12(4):979–1006. doi: 10.1137/S1052623499350013.

N. I. M. Gould, M. E. Hribar, and J. Nocedal (2001). On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM Journal on Scientific Computing*, 23(4):1376–1395.

N. I. M. Gould, D. Orban, and P. L. Toint (2003). CUTEr and SifDec, a Constrained and Unconstrained Testing Environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394. doi: 10.1145/962437.962439.

C. Greif, E. Moulding, and D. Orban (2014). Bounds on the eigenvalues of block matrices arising from interior-point methods. *SIAM Journal on Optimization*, 24(1):49–83. doi: 10.1137/120890600.

A. Griewank and A. Walther (2002). On Constrained Optimization by Adjoint based Quasi-Newton Methods. *Optimization Methods and Software*, 17:869–889. doi: 10.1080/1055678021000060829.

R. T. Haftka and M. P. Kamat (1989). Simultaneous Nonlinear Structural Analysis and Design. *Computational Mechanics*, 4:409–416. doi: 10.1007/BF00293046.

J. E. Hicken (2014). Inexact Hessian-vector products in reduced-space differential-equation constrained optimization. *Optimization and Engineering*, 15:575–608. doi: 10.1007/s11081-014-9258-6.

G. J. Kennedy and J. R. R. A. Martins (2013). A laminate parametrization technique for discrete ply angle problems with manufacturing constraints. *Structural and Multidisciplinary Optimization*, 48(2):379–393. doi: 10.1007/s00158-013-0906-9.

G. J. Kennedy and J. R. R. A. Martins (2014). A parallel finite-element framework for large-scale gradient-based design optimization of high-performance structures. *Finite Elements in Analysis and Design*, 87:56–73. doi: 10.1016/j.finel.2014.04.011.

G. J. Kennedy and J. R. R. A. Martins (2015). A parallel aerostructural optimization framework for aircraft design studies. *Structural and Multidisciplinary Optimization*. doi: 10.1007/s00158-014-1108-9. (In press).

G. K. W. Kenway, G. J. Kennedy, and J. R. R. A. Martins (2014). Scalable parallel approach for high-fidelity steady-state aeroelastic analysis and derivative computations. *AIAA Journal*, 52(5):935–951. doi: 10.2514/1.J052255.

G. K. W. Kenway and J. R. R. A. Martins (2014). Multipoint high-fidelity aerostructural optimization of a transport aircraft configuration. *Journal of Aircraft*, 51(1):144–160. doi: 10.2514/1.C032150.

C.-J. Lin and J. J. Moré (1998). Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9:1100–1127. doi: 10.1137/S1052623498345075.

Z. Lyu, G. K. Kenway, and J. R. R. A. Martins (2015). Aerodynamic shape optimization studies on the Common Research Model wing benchmark. *AIAA Journal*. doi: 10.2514/1.J053318. (In press).

Z. Lyu and J. R. R. A. Martins (2014). Aerodynamic design optimization studies of a blended-wing-body aircraft. *Journal of Aircraft*, 51(5):1604–1617. doi: 10.2514/1.C032491.

H. J. Martínez (1988). Local and superlinear convergence of structured secant methods for the convex class. Technical report, Rice University, Houston, USA.

J. R. R. A. Martins and A. B. Lambe (2013). Multidisciplinary design optimization: A survey of architectures. *AIAA Journal*, 51(9):2049–2075. doi: 10.2514/1.J051895.

J. J. Moré and G. Toraldo (1989). Algorithms for bound constrained quadratic programming problems. *Numerische Mathematik*, 55:377–400. doi: 10.1007/BF01396045.

J. J. Moré and G. Toraldo (1991). On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1):93–113. doi: 10.1137/0801008.

T. Munson, J. Sarich, S. Wild, S. Benson, and L. C. McInnes (2012). TAO 2.0 Users Manual. Technical Report ANL/MCS-TM-322, Mathematics and Computer Science Division, Argonne National Laboratory. http://www.mcs.anl.gov/tao.

B. A. Murtagh and M. A. Saunders (1978). Large-scale linearly constrained optimization. *Mathematical Programming*, 14(1):41–72. doi: 10.1007/BF01588950.

B. A. Murtagh and M. A. Saunders (2003). MINOS 5.51 user's guide. Technical Report SOL 83-20R, Stanford University, STANFORD, CALIFORNIA, USA.

J. Nocedal and S. J. Wright (2006). *Numerical Optimization*. Springer, New York, 2nd edition.

J. Nocedal and Y. Yuan (1998). Combining trust region and line search techniques. Technical report, Advances in Nonlinear Programming, (Kluwer).

D. Orban (2014). NLPy — a large-scale optimization toolkit in Python. Cahier du GERAD G-2014-xx, GERAD, Montréal, QC, Canada. In preparation.

R. E. Perez, P. W. Jansen, and J. R. R. A. Martins (2012). pyOpt: a Python-based object-oriented framework for nonlinear constrained optimization. *Structural and Multidisciplinary Optimization*, 45(1):101–118. doi: 10.1007/s00158-011-0666-3.

N. M. K. Poon and J. R. R. A. Martins (2007). An adaptive approach to constraint aggregation using adjoint sensitivity analysis. *Structural and Multidisciplinary Optimization*, 34:61–73. doi: 10.1007/s00158-006-0061-7.

R. T. Rockafellar (1973). The multiplier method of Hestenes and Powell applied to convex programming. *Journal of Optimization Theory and Applications*, 12:555–562. doi: 10.1007/BF00934777.

S. Schlenkrich, A. Griewank, and A. Walther (2010). On the local convergence of adjoint Broyden methods. *Mathematical Programming*, 121:221–247. doi: 10.1007/s10107-008-0232-y.

L. A. Schmit (1960). Structural Design by Systematic Synthesis. In *2nd Conference on Electronic Computation*, pages 105–132, New York, NY. ASCE.

P. L. Toint (1997). Non-monotone trust-region algorithms for nonlinear optimization subject to convex constraints. *Mathematical Programming*, 77(3):69–94. doi: 10.1007/BF02614518.

M. J. Turner, R. W. Clough, H. C. Martin, and L. J. Topp (1956). Stiffness and deflection analysis of complex structures. *Journal of Aeronautical Science*, 23(9):805–823.

A. Wächter and L. T. Biegler (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57. doi: 10.1007/s10107-004-0559-y.

CHAPITRE 5 ARTICLE 2 : A REGULARIZED FACTORIZATION-FREE METHOD FOR EQUALITY-CONSTRAINED OPTIMIZATION

Sylvain Arreckx
École Polytechnique de Montréal & GERAD, Canada
Dominique Orban
École Polytechnique de Montréal & GERAD, Canada

Abstract

We propose a factorization-free method for equality-constrained optimization based on a problem in which all constraints are systematically regularized. The regularization is equivalent to applying an augmented Lagrangian method but the linear system used to compute a search direction is reminiscent of regularized sequential quadratic programming (SQP). A limited-memory BFGS approximation to second derivatives allows us to employ iterative methods for linear least squares to compute steps, resulting in a factorization-free implementation. We establish global and fast local convergence under weak assumptions. In particular, we do not require the LICQ and our method is suitable for degenerate problems. Numerical experiments show that our method significantly outperforms IPOPT with limited-memory BFGS approximations, which is a state-of-the-art implementation of SQP on equality-constrained problems. We include a discussion on generalizing our framework to other classes of methods and to problems with inequality constraints.

5.1 Introduction

We consider the general equality-constrained optimization problem

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to } \; c(x) = 0. \tag{5.1}$$

The main objective of this paper is to devise an implementable factorization-free algorithm for (5.1) in the large-scale case that is somewhat resilient to constraint degeneracy and does not require matrix-vector products with $J(x)^T J(x)$, where J is the Jacobian of c, as is the case with a standard augmented Lagrangian method. We propose a framework inspired by that of (Armand et al., 2012) in which all constraints are systematically regularized. We show

Ce chapitre correspond à un article soumis dans SIAM Journal on Optimization.

that the regularization can be interpreted as a proximal-point Hestenes-Powell augmented Lagrangian method applied to (5.1) in the same vein as Rockafellar (1976). Our method uses the proximal augmented Lagrangian as merit function to promote global convergence and asymptotically blends into a stabilized SQP method possessing fast local convergence properties. Thanks to appropriate limited-memory BFGS approximations of the Hessian of the Lagrangian, the linear system encountered at each iteration is symmetric and quasi-definite (SQD) (Vanderbei, 1995), permitting inexact solves and an entirely factorization-free implementation suggested by methods described by Arioli and Orban (2013). We assume that $f : \mathbb{R}^n \to \mathbb{R}$ and $c : \mathbb{R}^n \to \mathbb{R}^m$ are twice continuously differentiable, although we only use exact second derivatives as an instrument in the analysis and in numerical illustration. In practice, only first derivatives are required when employing L-BFGS approximations.

The Karush-Kuhn-Tucker (KKT) conditions for (5.1) are only necessary for optimality when a constraint qualification holds. The most widely used constraint qualification condition, the Linear Independence Constraint Qualification (LICQ), requires that all constraint gradients be linearly independent at a stationary point. When such a constraint qualification fails to hold, the KKT conditions cease to be reliable for optimality. When it fails to hold at intermediate iterates, computational difficulties also arise. In particular, the linear systems used to compute search directions may become singular. Our regularization scheme is designed so as to overcome such complications.

We show that our method possesses global convergence properties similar to those of augmented Lagrangian methods (Bertsekas, 1996; Birgin and Martínez, 2014). In addition, we show that whenever the sequence of iterates converges to an isolated minimizer, the algorithm reduces asymptotically to pure stabilized SQP iterations and converges superlinearly. The convergence rate is quadratic if second-derivative approximations and steps are sufficiently accurate. Our numerical experiments show that the proposed scheme is efficient and robust, even when solving problems for which the LICQ fails to hold at the solution or at intermediate iterates, and compares very favorably to IPOPT (Wächter and Biegler, 2006).

Related Work

Sequential quadratic programming (SQP) methods (Boggs and Tolle, 1995; Wilson, 1963) are among the most successful methods for the solution of (5.1). They compute steps via a sequence of subproblems in which a quadratic model of the Lagrangian is minimized subject to linearized constraints. Convergence is enforced by requiring an improvement in a merit function at each step. Each iteration of an SQP method requires the solution of a linear system that involves the constraint Jacobian and its transpose. Most convergence analyses for SQP

and most SQP implementations require that those linear systems be solved exactly. In many large-scale applications, constraint Jacobians are only available as linear operators. In such cases, systems must be solved iteratively and inexactly, and it is crucial to account for this inexactness in the design and convergence analysis. An inexact trust-region SQP algorithm for equality constrained optimization is introduced in Heinkenschloss and Ridzal (2014). A composite-step approach is described in which the step in decomposed into a quasi-normal and a tangential step. A set of stopping criteria designed for controlling the inexactness of substep computations is given and ensures global convergence of their algorithm. Byrd et al. (2009) propose an inexact line-search SQP method for (5.1) where steps are computed from an inexact solution of a KKT system. A perturbation of the Hessian of the Lagrangian is employed to deal with nonconvexity. The perturbation is determined iteratively and may require repeated KKT solves per step computation. Two distinct termination tests control the level of inexactness in the step computation procedure.

Stabilized SQP methods were designed to remedy the numerical and theoretical difficulties associated with degenerate problems (Fernàndez and Solodov, 2010; Hager, 1999; Wright, 2005). The term *stabilized* refers to the calming effect on multiplier estimates for degenerate problems (Hager, 1999; Wright, 1998). Stabilized SQP promises superlinear local convergence under certain assumptions, but not global convergence to a stationary point. Few globalizations of the local stabilized SQP scheme have been proposed so far. Fernández et al. (2012) combine stabilized SQP with the inexact restoration method to ensure convergence from an arbitrary starting point. Gill and Robinson (2013) establish connections between stabilized SQP and augmented Lagrangian methods, including primal-dual variants of the augmented Lagrangian. Izmailov et al. (2015) combine stabilized SQP with the usual augmented Lagrangian algorithm when inequalities are present. However, their algorithm doesn't allow the use of quasi-Newton approximations to second-order derivatives and the linear systems involved during optimization must be solved exactly. Armand and Omheni (2015) propose a primal-dual augmented Lagrangian approach to solve equality constrained optimization problems that is quadratically convergent. However, convergence assumes the LICQ, exact linear system solves and exact second derivatives.

Notation

Throughout the paper, $\|\cdot\|$ denotes the Euclidean norm and I denotes the identity matrix of appropriate size. For any symmetric and positive definite matrix H, the H-norm is defined as $\|u\|_H^2 := u^T H u$. We use $\lambda_{\min}(M)$ and $\lambda_{\max}(M)$ to denote the smallest and largest eigenvalue of any symmetric matrix M. Similarly, $\sigma_{\min}(A)$ and $\sigma_{\max}(A)$ denote the smallest and largest

singular values of any matrix A. For two non-negative scalar sequences a_k and b_k converging to zero, we use the Landau symbols $a_k = o(b_k)$ if $\lim_{k\to+\infty} a_k/b_k = 0$ and $a_k = \omega(b_k)$ if $b_k = o(a_k)$. We write $a_k = O(b_k)$ if there exists a constant $C > 0$, such that $a_k \leq Cb_k$ for large k and $a_k = \Theta(b_k)$ if $a_k = O(b_k)$ and $b_k = O(a_k)$.

The rest of the paper is organized as follows. Section 5.2 summarizes the connexion between augmented Lagrangians and regularized SQP methods. In Section 5.3, we describe our algorithm in detail. Its global convergence properties are given in Section 5.4. Local convergence is analyzed in Section 5.5. We describe our implementations and report on numerical experience in Section 5.6. We conclude and discuss extensions to our framework in Section 5.7.

5.2 A Primal-Dual Regularization and Regularized SQP Methods

The Lagrangian for (5.1) is defined as

$$L(x, y) := f(x) - c(x)^T y, \tag{5.2}$$

where $y \in \mathbb{R}^m$ is the vector of Lagrange multipliers associated to the equality constraints. If x^* is a local minimizer of (5.1), the KKT conditions require that there exist y^* such that

$$g(x^*) - J(x^*)^T y^* = 0, \quad c(x^*) = 0, \tag{5.3}$$

where $g(x) := \nabla f(x)$ and $J(x)$ is the Jacobian of $c(x)$. Existence of such a y^* is only guaranteed provided a constraint qualification condition holds at x^*. Should constraint qualifications fail to hold at x^*, there may exist no y^* satisfying (5.3) or there may exist an unbounded set of them (Gauvin, 1977). In either case, numerical methods, such as SQP methods, may be confronted with degenerate direction-finding subproblems.

For the purposes of this paper, we say that (5.1) is *degenerate* at a feasible x if the LICQ fails to hold at x, i.e., the vectors $\nabla c_i(x)$, $i = 1, \ldots, m$, are linearly dependent.

Consider applying an augmented Lagrangian method to (5.1). If we denote y_k the current approximation of the Lagrange multipliers, the k-th subproblem has the form

$$\min_{x\in\mathbb{R}^n} \; L(x, y_k) + \tfrac{1}{2}\delta_k^{-1}\|c(x)\|^2, \tag{5.4}$$

where $\delta_k > 0$ is a penalty parameter. Following the procedure outlined by Friedlander and

Orban (2012), it is not difficult to see that (5.4) may equivalently be written as

$$\min_{x\in\mathbb{R}^n, u\in\mathbb{R}^m} \; f(x) + \tfrac{1}{2}\delta_k\|u + y_k\|^2 \quad \text{subject to } \; c(x) + \delta_k u = 0, \tag{5.5}$$

for some new variables u. The problem (5.5) provides an interpretation of the augmented Lagrangian method as an adaptive constraint regularization process. Since the regularization acts on the constraints and adds a term to the objective involving the multipliers, we term it *dual*. Of paramount importance is the fact that the LICQ is satisfied at every feasible point of (5.5).

In addition to the dual regularization term, we follow Friedlander and Orban (2012) and add primal regularization in the form of a proximal-point term, so that the k-th subproblem takes the form

$$\min_{x\in\mathbb{R}^n, u\in\mathbb{R}^m} \; f(x) + \tfrac{1}{2}\rho_k\|x - x_k\|^2 + \tfrac{1}{2}\delta_k\|u + y_k\|^2 \quad \text{subject to } \; c(x) + \delta_k u = 0, \tag{5.6}$$

for a primal regularization parameter $\rho_k \geq 0$, where x_k is the current primal iterate.

The KKT conditions for (5.6),

$$\begin{bmatrix} g(x) + \rho_k(x - x_k) - J(x)^T y \\ \delta_k(u + y_k) - \delta_k y \\ c(x) + \delta_k u \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{5.7}$$

are therefore unconditionally necessary for optimality for any fixed value of $\rho_k \geq 0$ and $\delta_k > 0$. The following relationship between KKT points of (5.1) and those of (5.6) illustrates the fact that the primal-dual regularization is *exact*.

Theorem 7. *Suppose (x_k, u_k, y_k) is a KKT point of (5.6) for some $\rho_k \geq 0$ and $\delta_k > 0$. Then (x_k, y_k) is a KKT point of (5.1).*

Alternatively, suppose $\rho_k = 0$ and $(\bar{x}, \bar{u}, \bar{y})$ is a KKT point of (5.6) for some $\delta_k > 0$ and suppose $\bar{x}$ is feasible for (5.1). Then $\bar{u} = 0$, $\bar{y} = y_k$ and $(\bar{x}, \bar{y})$ is a KKT point of (5.1).

Conversely, suppose (x^*, y^*) is a KKT point of (5.1). Then $(x_k, 0, y_k) := (x^*, 0, y^*)$ is a KKT point of (5.6) for any $\rho \geq 0$ and $\delta > 0$.

Proof. Immediate, by direct comparison of (5.3) and (5.7). □

Sequential quadratic programming methods for (5.6) may be interpreted as applying Newton's

method to (5.7). A Newton-like step for (5.7) from (x_k, u_k, y_k) solves the linear system

$$\begin{bmatrix} H_k + \rho_k I & & -J_k^T \\ & \delta_k I & -\delta_k I \\ J_k & \delta_k I & \end{bmatrix} \begin{bmatrix} \Delta x \\ \Delta u \\ \Delta y \end{bmatrix} = - \begin{bmatrix} g_k - J_k^T y_k \\ \delta_k u_k \\ c_k + \delta_k u_k \end{bmatrix}, \tag{5.8}$$

where $g_k := g(x_k)$, $c_k := c(x_k)$, $J_k := J(x_k)$ and H_k is a symmetric approximation of $\nabla_{xx} L(x_k, y_k)$. The elimination of $\Delta u = -u_k + \Delta y$ yields the reduced system

$$\begin{bmatrix} H_k + \rho_k I & J_k^T \\ J_k & -\delta_k I \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = - \begin{bmatrix} g_k - J_k^T y_k \\ c_k \end{bmatrix}, \tag{5.9}$$

which is the familiar system encountered in stabilized SQP methods, e.g., (Wright, 1998), while the system used in classical SQP methods corresponds to $\delta_k = 0$. For simplicity in the rest of this paper, the coefficient matrix of (5.9) is referred as K_k. The system (5.9) may be interpreted as the KKT conditions of the quadratic subproblem

$$\begin{aligned} \min_{\Delta x, \Delta u} \quad & \nabla_x L(x_k, y_k)^T \Delta x + \tfrac{1}{2} \Delta x^T (H_k + \rho_k I) \Delta x + \tfrac{1}{2} \delta_k \|u_k + \Delta u\|^2 \\ \text{subject to} \quad & c_k + J_k \Delta x + \delta_k (u_k + \Delta u) = 0. \end{aligned} \tag{5.10}$$

Note that (5.10) itself always satisfies the LICQ and therefore infeasible subproblems never occur. The primal regularization term $\rho_k I$ may be interpreted as a convexifying term that encourages descent in an appropriate merit function.

Given a fixed $\bar{x} \in \mathbb{R}^n$, we define

$$\phi(x, y; \bar{x}, \rho, \delta) := f(x) - c(x)^T y + \tfrac{1}{2} \rho \|x - \bar{x}\|^2 + \tfrac{1}{2} \delta^{-1} \|c(x)\|^2. \tag{5.11}$$

For future reference, we note that

$$\nabla_x \phi(\bar{x}, y; \bar{x}, \rho, \delta) = \nabla_x L(\bar{x}, y) + \delta^{-1} J(\bar{x})^T c(\bar{x}) = g(\bar{x}) - J(\bar{x})^T (y - \delta^{-1} c(\bar{x})). \tag{5.12}$$

For simplicity of exposition, we write $\phi(x, y; \rho, \delta)$ instead of $\phi(x, y; x, \rho, \delta)$. We also let $w := (x, y)$ and define $F : \mathbb{R}^{n+m} \to \mathbb{R}^{n+m}$ as $F(w) := (\nabla_x L(w), c(x))$.

5.3 Main Algorithm

Algorithm 5.3.1 is a simplification of (Armand et al., 2012, Algorithm 1) that ignores inequality constraints and allows symmetric Hessian approximations. In the description, we make use of

the norm $\|F(w)\|_* := \|\nabla_x L(w)\| + \|c(x)\|$.

Algorithm 5.3.1 Outer iteration

1: Choose $\alpha \in (0,1)$, $\theta \in (0,1)$, and $\epsilon > 0$. Set $k = 0$.
2: If $\|F(w_k)\| < \epsilon$, terminate with final iterate w_k.
3: Choose a symmetric matrix H_k, $\rho_k \geq 0$, and $\delta_k \in [\min(\|F(w_k)\|, \alpha\delta_{k-1}), \delta_{k-1}]$. Compute a trial iterate w_k^+ as an approximate solution of

$$K_k(w_k^+ - w_k) + F(w_k) = 0. \tag{5.13}$$

4: Choose $\epsilon_k > 0$. If

$$\|F(w_k^+)\|_* \leq \theta\|F(w_k)\|_* + \epsilon_k, \tag{5.14}$$

then set $w_{k+1} = w_k^+$. Otherwise perform a sequence of inner iterations in order to find a new iterate w_{k+1} such that

$$\|F(w_{k+1})\|_* \leq \theta\|F(w_k)\|_* + \epsilon_k. \tag{5.15}$$

Increment k by one and return to Step 2.

The main idea of Algorithm 5.3.1 is to start each outer iteration with an extrapolation step (5.13). The extrapolation step is accepted if it achieves sufficient improvement in the first-order optimality residual. In the negative, an inner iteration procedure is started in order to identify an improved iterate.

A certain amount of flexibility is allowed in choosing the parameters ρ_k and δ_k at the beginning of each outer iteration. An important feature for global convergence is that it is allowed to keep δ_k fixed, at least after a certain number of iterations. An important feature for fast local convergence is that it is allowed to select $\delta_k = \|F(w_k)\|$ when close to an isolated minimizer.

Algorithm 5.3.2 describes the linesearch procedure used as the inner iteration, which is essentially a standard augmented Lagrangian subproblem solve in which steps are computed using the augmented form (5.16) followed by a Wolfe linesearch on the proximal augmented Lagrangian. During the inner iterations, y_k is kept fixed. The inner primal iterates and regularization parameter corresponding to the k-th outer iteration are denoted $x_{k,j}$, $\rho_{k,j}$ and $\delta_{k,j}$ for $j \geq 0$. The inner iterations stop as soon as the first-order optimality residual of (5.1) has sufficiently decreased. As in the standard augmented Lagrangian, the penalty parameter is decreased when dual feasibility improved but primal feasibility lags behind.

In Step 3 of Algorithm 5.3.1 and Step 3 of Algorithm 5.3.2, the linear system could be solved exactly, but there is flexibility to compute an inexact solution. The inexactness in the outer iteration is a departure from the framework of Armand et al. (2012).

In Algorithm 5.3.2, steplengths are computed so as to satisfy the Wolfe conditions. The reason

Algorithm 5.3.2 Inner iteration

1: Set j to 0. Choose an initial guess $w_{k,0}$ and $\delta_{k,0} > 0$. Choose c_1 and c_2 such that $0 < c_1 < c_2 < 1$.

2: If $\|\nabla_x L(x_{k,j}, y_k - \delta_{k,j}^{-1} c(x_{k,j}))\| \leq \theta \|\nabla_x L(x_{k,0}, y_k)\| + \frac{1}{2}\epsilon_k$, then
if $\|c(x_{k,j})\| \leq \theta \|c(x_{k,0})\| + \frac{1}{2}\epsilon_k$, stop with $w_{k+1} = (x_{k,j}, y_k - \delta_{k,j}^{-1} c(x_{k,j}))$,
otherwise, set $\delta_{k,j} = \delta_{k,j-1}/10$.
Go to Step 3.

3: Choose a symmetric matrix $H_{k,j}$ and $\rho_{k,j} \geq 0$. Compute Δx_j as an approximate solution to

$$\begin{bmatrix} H_{k,j} + \rho_{k,j} I & J_{k,j}^T \\ J_{k,j} & -\delta_{k,j} I \end{bmatrix} \begin{bmatrix} \Delta x_j \\ -\Delta y_j \end{bmatrix} = - \begin{bmatrix} g_{k,j} - J_{k,j}^T y_k \\ c_{k,j} \end{bmatrix}. \tag{5.16}$$

4: Set $x_{k,j+1} = x_{k,j} + \alpha_j \Delta x_j$, where α_j is obtained using a line search and satisfies the Wolfe conditions:

$$\phi(x_{k,j+1}, y_k; x_{k,j}, \rho_{k,j}, \delta_{k,j}) \leq \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j}) + c_1 \alpha_j \nabla\phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})^T \Delta x_j$$
$$\nabla\phi(x_{k,j+1}, y_k; x_{k,j}, \rho_{k,j}, \delta_{k,j})^T \Delta x_j \geq c_2 \nabla\phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})^T \Delta x_j.$$

Increment j by one and return to Step 2.

for this requirement is that the global convergence analysis is based on a simple application of Zoutendijk's theorem (Nocedal and Wright, 2006, Theorem 3.2), which requires the Wolfe conditions. We believe it is possible to develop a global convergence analysis based solely on the Armijo condition, in the vein of Armand and Omheni (2015).

The next section examines the global convergence properties of Algorithms 5.3.1 and 5.3.2 and, in particular, what conditions should be imposed on inexact steps.

5.4 Global Convergence

In this section, we examine in turn the convergence of Algorithms 5.3.1 and 5.3.2.

5.4.1 Convergence of the Inner Iterations

The convergence of the inner iterations is divided into two parts depending on whether linear systems are solved exactly or not. In Section 5.4.1, we assume that the linear systems of Algorithms 5.3.1 and 5.3.2 are solved exactly. That situation covers the case where the systems are solved via a factorization, whether exact second derivatives are used or not. It also covers the case where the linear systems are solved iteratively but with an extremely tight tolerance, although that is not realistic in practice. In Section 5.4.1, we assume that systems are solved inexactly and we describe the iterative procedure that we use. The latter relies on H_k being

positive definite and such that linear systems with coefficient H_k can be solved easily and cheaply. Such a situation occurs when H_k is a limited-memory BFGS approximation to the second derivatives, and that is our selection of choice. With that choice, H_k itself is never really needed and we simply maintain its inverse implicitly (Liu and Nocedal, 1989). Other choices are of course possible, including $H_k = I$, or a positive-definite diagonal approximation of $\nabla^2_{xx} L(x_k, y_k)$.

In this section, k denotes the outer iteration index and appears everywhere to avoid ambiguity. We refer to the coefficient of the linear system at outer iteration k and inner iteration j in Step 3 of Algorithm 5.3.2 as $K_{k,j}$.

Our main working assumption is as follows.

Assumption 1. *The gradients $g_{k,j}$, the matrices $H_{k,j}$ and the matrices $J_{k,j}$ are uniformly bounded for all $j \in \mathbb{N}$. Moreover, $\rho_{k,j}$ is bounded for all $j \in \mathbb{N}$.*

Exact System Solves

When $H_{k,j}$ is not positive definite, which is typically the case when exact second-derivatives are used, $\rho_{k,j}$ must be sufficiently large to ensure that Δx_j is a descent direction for the proximal augmented Lagrangian. Linear systems may be solved using a symmetric indefinite factorization such as the multifrontal implementation MA57 of Duff (2004). Because this factorization reveals the inertia of $K_{k,j}$, the regularization parameter $\rho_{k,j}$ can be increased until the correct inertia is detected (Gould, 1985). Such a procedure is similar to that used in IPOPT (Wächter and Biegler, 2006). Those observations motivate the following assumption.

Assumption 2. *The matrices $H_{k,j} + \rho_{k,j} I + \delta_{k,j}^{-1} J_{k,j}^T J_{k,j}$ are uniformly positive definite and uniformly bounded for all $j \in \mathbb{N}$, i.e., there exist constants $\overline{\sigma} \geq \underline{\sigma} > 0$ such that for all $j \in \mathbb{N}$ and all $d \in \mathbb{R}^n$,*

$$\underline{\sigma} \|d\|^2 \leq d^T (H_{k,j} + \rho_{k,j} I + \delta_{k,j}^{-1} J_{k,j}^T J_{k,j}) d \leq \overline{\sigma} \|d\|^2.$$

Assumption 2 implies that $\{H_{k,j} + \rho_{k,j} I\}$ is uniformly positive definite over the nullspace of $J_{k,j}$ for all $j \in \mathbb{N}$.

Theorem 8 (Inner iteration, exact solves)**.** *Suppose that Assumption 2 holds and that $\phi(\cdot, y_k; \rho_{k,j}, \delta_{k,j})$ is bounded below for all j. Then Algorithm 5.3.2 generates a sequence of iterates $x_{k,j}$ such that*

$$\lim_{j \to +\infty} \|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\| = 0.$$

Proof. We eliminate Δy_j from (5.16) and use (5.12) to obtain

$$\left(H_{k,j} + \rho_{k,j} I + \delta_{k,j}^{-1} J_{k,j}^T J_{k,j}\right) \Delta x_j = -\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j}). \tag{5.17}$$

We take the inner product of both sides of (5.17) with Δx_j and use Assumption 2, and obtain

$$-\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})^T \Delta x_j \geq \underline{\sigma} \|\Delta x_j\|^2. \tag{5.18}$$

Assumption 2 and (5.17) yield

$$\begin{aligned}
\|\Delta x_j\| &\geq \lambda_{\min}\left((H_{k,j} + \rho_{k,j} I + \delta_{k,j}^{-1} J_{k,j}^T J_{k,j})^{-1}\right) \|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\| \\
&= \left(\lambda_{\max}(H_{k,j} + \rho_{k,j} I + \delta_{k,j}^{-1} J_{k,j}^T J_{k,j})\right)^{-1} \|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\| \\
&\geq \overline{\sigma}^{-1} \|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\|.
\end{aligned}$$

Therefore

$$-\frac{\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})^T \Delta x_j}{\|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\| \, \|\Delta x\|} \geq \underline{\sigma}/\overline{\sigma} > 0.$$

Zoutendijk's theorem ensures that

$$\lim_{j \to \infty} \left(-\frac{\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})^T \Delta x}{\|\nabla \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\| \|\Delta x\|}\right)^2 \|\nabla \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\|^2 = 0,$$

from which the desired result follows immediately. □

Theorem 8 implies that the first stopping condition of Algorithm 5.3.1 is satisfied after a finite number of iterations because (5.12) can also be written

$$\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j}) = \nabla_x L(x_{k,j}, y_k - \delta_{k,j}^{-1} c(x_{k,j})).$$

In order to determine when the second stopping condition holds, we consider the following assumption.

Assumption 3. *The sequence $\{\delta_{k,j}\}_{j \in \mathbb{N}}$ is bounded away from zero.*

If Assumption 3 holds, the mechanism of Algorithm 5.3.2 guarantees that the stopping condition $\|c(x_{k,j})\| \leq \theta \|c(x_{k,0})\| + \frac{1}{2}\epsilon_k$ is eventually satisfied as well.

If Assumption 3 fails, there is an index set $\mathcal{J}$ such that $\lim_{j \in \mathcal{J}} \delta_{k,j} = 0$. In the situation where

$\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})$ remains bounded, we have

$$\begin{aligned} 0 &= \lim_{j \in \mathcal{J}} \delta_{k,j} \nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j}) \\ &= \lim_{j \in \mathcal{J}} \delta_{k,j} \left(g(x_{k,j}) - J(x_{k,j})^T y_k \right) + J(x_{k,j})^T c(x_{k,j}). \end{aligned}$$

Under Assumption 1, the first term of the previous left-hand side converges to zero, and therefore

$$\lim_{j \in \mathcal{J}} J(x_{k,j})^T c(x_{k,j}) = 0.$$

In other words, there is a limit point of the sequence generated by Algorithm 5.3.2 that is stationary for the (typically underdetermined) least-squares problem

$$\min_{x \in \mathbb{R}^n} \ \tfrac{1}{2} \|c(x)\|^2.$$

This situation occurs when the augmented Lagrangian multiplier estimates $y_k - \delta_{k,j}^{-1} c(x_{k,j})$ remain bounded, which is a common assumption in the convergence analysis of augmented Lagrangian methods. Multiplier unboundedness may be an indication that the constraints are not linearly independent.

Inexact System Solves: a Quasi-Newton Strategy

A difficulty associated with the iterative solution of systems of the form (5.16), e.g., using MINRES (Paige and Saunders, 1975), is the need to balance residuals associated to each block equation, as in (Byrd et al., 2009). The framework detailed in this section solves one of the block equations exactly while controlling the residual associated to the other. That is done by transforming (5.16) into the first-order optimality conditions of a preconditioned linear least-squares problem. The preconditioner used, $H_{k,j}^{-1}$ in the present case, must be applied at each iteration of an iterative method for least-squares problems. It is therefore crucial that $H_{k,j}^{-1}$ be positive definite and cheaply applicable.

In this section, we make the following assumption.

Assumption 4. *The matrices $H_{k,j}$ are uniformly positive definite for all $j \in \mathbb{N}$.*

For the above reasons, and in order to preserve hope for fast local convergence when close to an isolated minimizer, we set $H_{k,j}$ to a limited-memory BFGS approximation of the Hessian of the Lagrangian (Liu and Nocedal, 1989) that is possibly modified to take into account the fact that the Hessian of the Lagrangian cannot be expected to be positive definite. Details are not relevant here and will be given in Section 5.6. By construction, L-BFGS approximations are

always positive definite. In addition, their inverse can be implicitly maintained and updated along the iterations, and can be applied to a vector cheaply using either the two-loop recursion or the compact storage format (Byrd et al., 1994), or by maintaining the approximation in factored form (Dennis and Schnabel, 1996, Algorithm A9.4.2).

With such a choice, $K_{k,j}$ is SQD, and therefore nonsingular irrespective of the rank of $J_{k,j}$. We always set $\rho_{k,j} = 0$ when $H_{k,j}$ is a L-BFGS approximation.

We first cast (5.16) as a least-square problem. We introduce $\Delta\bar{y} := \Delta y + \delta_{k,j}^{-1} c_{k,j}$, and rewrite (5.16) equivalently as

$$\begin{bmatrix} H_{k,j} & J_{k,j}^T \\ J_{k,j} & -\delta_{k,j} I \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta\bar{y} \end{bmatrix} = \begin{bmatrix} b_{k,j} \\ 0 \end{bmatrix} \tag{5.19}$$

where $b_{k,j} = -g_{k,j} + J_{k,j}^T(y_k - \delta_{k,j}^{-1} c_{k,j}) = -\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})$. Birgin and Martínez (2014) use a similar system obtained from linear algebra transformations of the Newton equations $\nabla_{xx}^2 \phi(x, y; \rho) d = -\nabla_x \phi(x, y; \rho)$.

The shifted system (5.19) can be seen as the necessary and sufficient optimality conditions of the regularized and preconditioned least-squares problem

$$\min_{\Delta\bar{y}} \ \tfrac{1}{2} \left\| J_{k,j}^T \Delta\bar{y} + b_{k,j} \right\|_{H_{k,j}^{-1}}^2 + \tfrac{1}{2} \|\Delta\bar{y}\|_{\delta_{k,j} I}^2. \tag{5.20}$$

The latter can be solved approximately using a stopping criterion based exclusively on the residual of the second block equation of (5.19), i.e., the optimality residual of (5.20):

$$r_{k,j} := J_{k,j} \Delta x + \delta_{k,j} \Delta\bar{y}, \tag{5.21}$$

while the least-squares residual is $\Delta x := H_{k,j}^{-1}(J_{k,j}^T \Delta\bar{y} + b_{k,j})$.

Our choice is to solve (5.20) with the LSMR method of Fong and Saunders (2011) modified as recommended by Arioli and Orban (2013) to accommodate non-Euclidean norms. This choice is motivated by the fact that we wish to reduce the residual of (5.19), or equivalently, of (5.16), below a certain threshold. The least-squares interpretation guarantees that the first block equation is always satisfied exactly, by definition of the least-squares residual. The residual of the second block equation is precisely $r_{k,j}$. An important property of LSMR, that is not shared by other methods such as LSQR (Paige and Saunders, 1982), is that it decreases the norm of the optimality residual of (5.20), i.e., the norm of $r_{k,j}$, monotonically. Finally, Arioli and Orban (2013) show that using LSMR as described above can save half of the iterations as compared to using MINRES with the natural preconditioner blkdiag($H_{k,j}$, $\delta_{k,j} I$).

Our implementation uses two termination tests. The first guarantees sufficient descent.

Termination Test 1. *Let $\{\gamma_j\}$ be any positive sequence that is bounded away from zero. A step $(\Delta x, \Delta\bar{y})$ is an acceptable inexact solution of* (5.19) *if*

$$\|r_{k,j}\|^2_{\delta^{-1}_{k,j}I} + \gamma_j \|b_{k,j}\|^2_{H^{-1}_{k,j}} \leq \|J^T_{k,j}\Delta\bar{y} + b_{k,j}\|^2_{H^{-1}_{k,j}} + \|\Delta\bar{y}\|^2_{\delta_{k,j}I}. \tag{5.22}$$

We defer comments on Termination Test 1 until the end of this section.

Lemma 1. *Let Assumptions 1 and 4 be satisfied. Suppose that Termination Test 1 is satisfied. Then there exists a constant $\gamma > 0$ such that*

$$-\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})^T \Delta x \geq \gamma \, \|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\|^2.$$

Proof. Let us denote $\phi_{k,j} := \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})$ and $M_{k,j} := H_{k,j} + \delta^{-1}_{k,j} J^T_{k,j} J_{k,j}$ for conciseness. The first block equation of (5.19) and (5.21) yield

$$\begin{aligned} -\nabla_x \phi^T_{k,j} \Delta x &= \Delta x^T H_{k,j} \Delta x - \Delta\bar{y}^T J_{k,j} \Delta x \\ &= \Delta x^T H_{k,j} \Delta x + \delta_{k,j} \Delta\bar{y}^T \Delta\bar{y} - \Delta\bar{y}^T r_{k,j}. \end{aligned} \tag{5.23}$$

Isolating $\Delta\bar{y}$ in the second block equation of (5.19), substituting it into the first one and using (5.21) gives

$$-\nabla_x \phi_{k,j} = M_{k,j}\Delta x - \delta^{-1}_{k,j} J^T_{k,j} r_{k,j}. \tag{5.24}$$

Thus

$$-\nabla_x \phi^T_{k,j} \Delta x = \Delta x^T M_{k,j} \Delta x - \delta^{-1}_{k,j} \Delta x^T J^T_{k,j} r_{k,j}. \tag{5.25}$$

Adding (5.23) and (5.25) together, dividing by 2 and noting that the norm of the residual $r_{k,j}$ can be expressed as

$$\|J_{k,j}\Delta x + \delta_{k,j}\Delta\bar{y}\|^2 := r^T_{k,j} r_{k,j} = \Delta x^T J^T_{k,j} r_{k,j} + \delta_{k,j} \Delta\bar{y}^T r_{k,j}, \tag{5.26}$$

leads to

$$\begin{aligned} -\nabla_x \phi^T_{k,j} \Delta x &= \tfrac{1}{2}\Delta x^T M_{k,j} \Delta x + \tfrac{1}{2}\Delta x^T H_{k,j} \Delta x + \tfrac{1}{2}\|\Delta\bar{y}\|^2_{\delta_{k,j}I} - \tfrac{1}{2}\|r_{k,j}\|^2_{\delta^{-1}_{k,j}I} \\ &= \tfrac{1}{2}\Delta x^T M_{k,j} \Delta x + \tfrac{1}{2}\|J^T_{k,j}\Delta\bar{y} + b_{k,j}\|^2_{H^{-1}_{k,j}} + \tfrac{1}{2}\|\Delta\bar{y}\|^2_{\delta_{k,j}I} - \tfrac{1}{2}\|r_{k,j}\|^2_{\delta^{-1}_{k,j}I} \\ &\geq \tfrac{1}{2}\|J^T_{k,j}\Delta\bar{y} + b_{k,j}\|^2_{H^{-1}_{k,j}} + \tfrac{1}{2}\|\Delta\bar{y}\|^2_{\delta_{k,j}I} - \tfrac{1}{2}\|r_{k,j}\|^2_{\delta^{-1}_{k,j}I} \\ &\geq \tfrac{1}{2}\|J^T_{k,j}\Delta\bar{y} + b_{k,j}\|^2_{H^{-1}_{k,j}} + \tfrac{1}{2}\|\Delta\bar{y}\|^2_{\delta_{k,j}I} - \tfrac{1}{2}\|r_{k,j}\|^2_{\delta^{-1}_{k,j}I}. \end{aligned}$$

Using (5.22) yields $-\nabla_x \phi_{k,j}^T \Delta x \geq \frac{1}{2}\gamma_j \|\nabla_x \phi_{k,j}\|^2_{H_{k,j}^{-1}} \geq \gamma \|\nabla_x \phi_{k,j}\|^2$, where $\gamma := \frac{1}{2}\inf_j \gamma_j / \sup_j \lambda_{\max}(H_{k,j}) > 0$. □

The second termination test is standard in LSMR.

Termination Test 2. *Let $\mu > 0$ and $0 \leq \beta_2 \leq 1$ be given constants. A step $(\Delta x, \Delta \bar{y})$ is an acceptable inexact solution of (5.19) if*

$$\|r_{k,j}\|_{\delta_{k,j}^{-1} I} \leq \mu \min(1, \delta_{k,j}^{\beta_2}) \|b_{k,j}\|_{H_{k,j}^{-1}}. \tag{5.27}$$

Termination Test 2 leads directly to convergence of the inner iterations.

Theorem 9 (Inner iteration, inexact solves)**.** *Suppose that Assumptions 1 and 4 hold, and that Termination Tests 1 and 2 are satisfied. Then Algorithm 5.3.2 generates a sequence of iterates $x_{k,j}$ such that*

$$\lim_{j \to \infty} \|\nabla_x \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})\| = 0.$$

Proof. We use the shorthands $\phi_{k,j} := \phi(x_{k,j}, y_k; \rho_{k,j}, \delta_{k,j})$ and $M_{k,j} := H_{k,j} + \delta_{k,j}^{-1} J_{k,j}^T J_{k,j}$ for conciseness. Assumption 4 implies that there exists $\kappa_H > 0$ such that $\|b_{k,j}\|_{H_{k,j}^{-1}} \leq \kappa_H \|b_{k,j}\|$ and $\kappa_M > 0$ such that $\|M_{k,j}^{-1}\| \leq \kappa_M$ for all j. From (5.24) and Termination Test 2, we have

$$\begin{aligned}
\|\Delta x\| &\leq \|M_{k,j}^{-1}\| \, \| - \nabla_x \phi_{k,j} + \delta_{k,j}^{-1} J_{k,j}^T r_{k,j}\| \\
&\leq \kappa_M \left(\|\nabla_x \phi_{k,j}\| + \delta_{k,j}^{-1} \|J_{k,j}^T r_{k,j}\| \right) \\
&\leq \kappa_M \left(\|\nabla_x \phi_{k,j}\| + \|J_{k,j}^T\| \, \|r_{k,j}\|_{\delta_{k,j}^{-1}} \right) \\
&\leq \kappa_M \left(\|\nabla_x \phi_{k,j}\| + \mu \min(1, \delta_{k,j}^{\beta_2}) \sigma_{\max}(J_{k,j}) \kappa_H \|\nabla_x \phi_{k,j}\| \right) \\
&\leq \kappa \, \|\nabla_x \phi_{k,j}\|
\end{aligned} \tag{5.28}$$

with $\kappa := \sup_j \, \kappa_M(1 + \mu \sigma_{\max}(J_{k,j}) \kappa_H) > 0$.

Lemma 1 and (5.28) together imply

$$-\frac{\nabla_x \phi_{k,j}^T \Delta x}{\|\nabla_x \phi_{k,j}\| \, \|\Delta x\|} \geq \frac{\gamma}{\kappa} > 0.$$

At this point, we are in position to apply Zoutendijk's theorem and conclude as in the proof of Theorem 8. □

In view of Theorem 9, it is easier to provide an interpretation of Termination Test 1. Firstly, under Assumption 3, the sequence $\{\gamma_j\}$ may be chosen as $\gamma_j := \kappa_1 \delta_{k,j}^{\kappa_2}$ for certain positive

constants κ_1 and κ_2. Secondly, the right-hand side of (5.22) is the objective value of (5.20). The first term in the left-hand side is the norm of (5.21), which represents the optimality residual of (5.20), and which decreases monotonically to zero along the LSMR iterations. The second term in the left-hand side is the norm of (5.12), which represents the first-order optimality conditions of the minimization of the augmented Lagrangian, and which approaches zero under the assumptions of Theorem 9. The role of the sequence $\{\gamma_j\}$ is to allow sufficient room for satisfaction of (5.22) even when the value of the right-hand side is small. Note however that the optimal value of the right-hand side can only be zero if $b_{k,j} = 0$, which means that $x_{k,j}$ is first-order stationary for the augmented Lagrangian.

As in Section 5.4.1, Assumption 3 ensures that the second stopping condition of Algorithm 5.3.2 is satisfied after a finite number of iterations.

5.4.2 Convergence of the Outer Iterations

We now analyze the global convergence of the outer iterations. In this section, we assume that Algorithm 5.3.2 succeeds in computing a new iterate w_{k+1} that satisfies (5.15) each time it is called at Step 4 of Algorithm 5.3.1. The following corollary results immediately from Theorems 8 and 9.

Corollary 1. *Assume Algorithm 5.3.2 succeeds each time it is called at Step 4 of Algorithm 5.3.1. Then Algorithm 5.3.1 generates iterates w_k such that*

$$\|F(w_{k+1})\|_* \leq \theta \|F(w_k)\|_* + \epsilon_k.$$

The following result, which is a direct consequence of (Armand et al., 2012, Theorem 3.3), describes the behavior of the sequence of outer iterates.

Theorem 10 (Outer iteration)**.** *Assume Algorithm 5.3.2 succeeds each time it is called at Step 4 of Algorithm 5.3.1 and that the sequence $\{\epsilon_k\}$ converges to zero. Then Algorithm 5.3.1 generates a sequence of iterates w_k such that $\{F(w_k)\}$ converges to zero.*

Proof. Let $\ell := \limsup_{k\to\infty} \|F(w_k)\|_*$. Taking the limit superior in (5.15) yields

$$\ell \leq \theta\ell + \limsup_{k\to+\infty} \epsilon_k.$$

Because $\lim_{k\to\infty} \epsilon_k = \limsup_{k\to+\infty} \epsilon_k = 0$, we have $\ell \leq \theta\ell$, i.e., $\ell = 0$, which means that $\{F(w_k)\}$ converges to zero. □

5.5 Local Convergence

In this section, we analyze the asymptotic behavior of $\{w_k\}$ under the assumption that it converges to a stationary point satisfying certain assumptions. We establish that the rate of convergence of $\{w_k\}$ is Q-superlinear provided δ_k asymptotically approaches zero sufficiently fast, which is ensured by selecting $\delta_k = \|F(w_k)\|$ in Algorithm 5.3.1. This last requirement is a departure from standard augmented Lagrangian methods and ensures the transition to a stabilized SQP method in the local regime.

The analysis in this section broadly follows that of Armand et al. (2012) and Wright (1998). The results differ in two important respects. Firstly, we allow quasi-Newton approximations to second-order derivatives. Secondly, we also allow inexact solutions to the extrapolation linear system (5.13).

Assume x^* is a stationary point of (5.1) that satisfies (5.3). We use $\mathcal{Y}$ to denote the set of Lagrange multipliers associated to x^*, i.e.,

$$\mathcal{Y} = \{y^* \in \mathbb{R}^m \mid (x^*, y^*) \text{ satisfies the KKT conditions (5.3)}\}.$$

We also use the notation $\mathcal{S} := \{x^*\} \times \mathcal{Y}$.

Because $\nabla_x L(x^*, \cdot)$ is linear, $\mathcal{Y}$ is a closed convex set. It is well known that $\mathcal{Y}$ is a singleton under the assumption that $J(x^*)$ has full row rank and may be unbounded or empty if that assumption fails to hold (Gauvin, 1977).

Our working assumptions for this section are as follows.

Assumption 5. *The sequence $\{w_k\}$ generated by Algorithm 5.3.1 converges to $w^* = (x^*, y^*)$ for a certain $y^* \in \mathcal{Y}$.*

Assumption 6. *The functions f and c are twice continuously differentiable with locally Lipschitz second derivatives on $\mathbb{R}^n$.*

Assumption 7. *δ_k is chosen as $\|F(w_k)\|$ at Step 3 of Algorithm 5.3.1.*

Assumption 8. *H_k is uniformly bounded, i.e., there exists $\kappa > 0$ such that $\|H_k\| \leq \kappa$ for all $k = 0, 1, \ldots$.*

When (5.13) is solved inexactly, Assumption 4 from the global regime is sufficient for our purposes. Whether (5.13) is solved exactly or not, we establish fast local convergence under the following, weaker, assumption.

Assumption 9. *The approximation H_k is sufficiently positive definite on the nullspace of $J(x^*)$ for all sufficiently large k, i.e., there exists $\eta > 0$, such that $z^T H_k z \geq \eta \|z\|^2$ for all $z \in \text{Null}(J(x^*))$.*

Assumption 9 implies that we may set $\rho_k = 0$ for all sufficiently large k.

The following assumption states that H_k is an increasingly accurate approximation of $\nabla^2_{xx} L(x^*, y^*)$ along Δx.

Assumption 10. *There exists $0 < \beta_1 \leq 1$ such that*

$$\|(H_k - \nabla^2_{xx} L(x^*, y^*))\Delta x\| = O(\|\Delta x\|^{1+\beta_1})$$

for all sufficiently large k, where Δx is computed from (5.13).

Assumption 10 is reminiscent of the Dennis and Moré (1977) condition in unconstrained optimization but is more demanding. Though it is unlikely that any quasi-Newton approximation satisfies Assumption 10 in the strong form given, the analysis below illustrates how the assumption allows us to specify the precise convergence rate of the sequence of iterates. It is more likely that a quasi-Newton approximation satisfies

$$(H_k - \nabla^2_{xx} L(x^*, y^*))\Delta x = o(\Delta x) \tag{5.29}$$

instead. As we comment at the end of the present section, the local convergence analysis holds under that weaker assumption, except that the exact convergence rate cannot be specified.

The global convergence analysis does not depend on whether, how, or how accurately we solve (5.13). The local analysis, however, depends on (5.13) crucially. In this section, we specify how accurate the step computation should be. Note that (5.13) can be shifted to least-squares form exactly as (5.19):

$$\begin{bmatrix} H_k & J_k^T \\ J_k & -\delta_k I \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta \bar{y} \end{bmatrix} = \begin{bmatrix} b_k \\ 0 \end{bmatrix}, \tag{5.30}$$

where $\Delta \bar{y} := \Delta y + \delta_{k,j}^{-1} c_{k,j}$ and $b_k = -\nabla_x \phi(x_k, y_k; 0, \delta_k)$. We use Termination Test 2 as our stopping condition. We repeat it here without mention of the index j.

Termination Test 3. *Let $\mu > 0$ and $0 \leq \beta_2 \leq 1$ be given constants. A step $(\Delta x, \Delta \bar{y})$ computed in an inexact solve of* (5.30) *at Step 3 of Algorithm 5.3.1 is acceptable if*

$$\|r_k\|_{\delta_k^{-1} I} \leq \mu \min(1, \delta_k^{\beta_2}) \|b_k\|_{H_k^{-1}}. \tag{5.31}$$

According to Theorem 10, $\delta_k \to 0$ and for all sufficiently large k, it realizes the minimum in (5.31).

For any $\epsilon > 0$, we define

$$\mathcal{N}(\epsilon) := \{(x, y) \mid \exists \bar{y} \in \mathcal{Y} \ \|(x, y) - (x^*, \bar{y})\| \leq \epsilon\}.$$

We denote by P the projection onto $\mathcal{Y}$, i.e.,

$$P(y) := \arg\min\{\|y - \bar{y}\| \mid \bar{y} \in \mathcal{Y}\},$$

which is well defined because $\mathcal{Y}$ is closed and convex. Finally, the Euclidean distance from (x, y) to $\mathcal{S}$ is denoted

$$\operatorname{dist}((x, y), \mathcal{S}) := \inf\{\|(x, y) - (x^*, \bar{y})\| \mid \bar{y} \in \mathcal{Y}\} = \|(x, y) - (x^*, P(y))\|.$$

For any $w = (x, y) \in \mathcal{N}(\epsilon)$, we use the notation δ to denote $\|F(w)\|$, in accordance with Assumption 7.

Lemma 2. *Suppose that Assumptions 6 and 7 hold. Then there exists a constant $\epsilon > 0$ such that for all $(x, y) \in \mathcal{N}(\epsilon)$ we have* $\operatorname{dist}((x, y), \mathcal{S}) = \Omega(\delta)$.

Proof. Let $\epsilon > 0$ be arbitrary and $(x, y) \in \mathcal{N}(\epsilon)$. It follows from (5.3) and Assumption 6 that

$$\|\nabla_x L(x, y)\| = \|\nabla_x L(x, y) - \nabla_x L(x^*, P(y))\| = O(\operatorname{dist}((x, y), \mathcal{S})).$$

Similarly,

$$\|c(x)\| = \|c(x) - c(x^*)\| = O(\|x - x^*\|) = O(\operatorname{dist}((x, y), \mathcal{S})).$$

Thus $\delta = O(\operatorname{dist}((x, y), \mathcal{S}))$. □

Wright (1998) establishes the converse of Lemma 2 under the Mangasarian and Fromovitz constraint qualification condition, which, in the case of equality constraints, amounts to the linear independence constraint qualification condition. Izmailov and Solodov (2012) establish a similar result without assuming a constraint qualification but by restricting attention to a neighborhood of (x^*, y^*). We include the proof for completeness. We denote $\mathcal{B}_\epsilon(x^*, y^*)$ the ball centered at (x^*, y^*) of radius $\epsilon > 0$.

Lemma 3. *Suppose that Assumptions 6, 7 and 9 hold. Then there exists $\epsilon > 0$ such that for all $(x, y) \in \mathcal{B}_\epsilon(x^*, y^*)$, we have* $\operatorname{dist}((x, y), \mathcal{S}) = O(\delta)$.

Proof. By contradiction, suppose that for any $\epsilon > 0$, there exists $(x, y) \in \mathcal{B}_\epsilon(x^*, y^*)$ such that $\delta = o(\|x - x^*\|)$ and $\delta = o(\|y - y^*\|)$. By selecting a sequence $\{\epsilon_k\} \to 0$, we determine sequences $\{x_k\} \to x^*$ and $\{y_k\} \to y^*$ such that $\delta_k = o(\|x_k - x^*\|)$ and $\delta_k = o(\|y_k - y^*\|)$.

With the purpose of deriving a contradiction with Assumptions 6 and 9, we use (5.3) and the fact that $\nabla_x L(x_k, y_k) = O(\delta_k) = o(\|x_k - x^*\|)$ by assumption to deduce

$$\begin{aligned}
\nabla^2_{xx} L(x^*, y^*)(x_k - x^*) &= \nabla_x L(x_k, y^*) - \nabla_x L(x^*, y^*) + o(\|x_k - x^*\|) \\
&= \nabla_x L(x_k, y_k) - J(x_k)^T (y_k - y^*) + o(\|x_k - x^*\|) \\
&= -J(x_k)^T (y_k - y^*) + o(\|x_k - x^*\|) \\
&= -J(x^*)^T (y_k - y^*) \\
&\quad - (J(x_k) - J(x^*))^T (y_k - y^*) + o(\|x_k - x^*\|) \\
&= -J(x^*)^T (y_k - y^*) + o(\|x_k - x^*\|). \qquad (5.32)
\end{aligned}$$

Similarly, our contradiction assumption gives

$$J(x^*)(x_k - x^*) = c(x_k) - c(x^*) + o(\|x_k - x^*\|) = o(\|x_k - x^*\|). \qquad (5.33)$$

Reducing to a subsequence if necessary, there exists a vector z with $\|z\| = 1$ such that $\{(x_k - x^*)/\|x_k - x^*\|\} \to z$. We take limits in (5.32) and (5.33), and obtain

$$\nabla^2_{xx} L(x^*, y^*) z \in \text{Range}(J(x^*)^T) \quad \text{and} \quad z \in \text{Null}(J(x^*)),$$

which contradicts Assumption 9. Thus there exists $\epsilon > 0$ such that for all $(x, y) \in \mathcal{B}_\epsilon(x^*, y^*)$, we have $\|x - x^*\| = O(\delta)$.

Let $(x, y) \in \mathcal{B}_\epsilon(x^*, y^*)$. The linear system $J(x^*)^T(y - \tilde{y}) = \nabla_x L(x^*, y)$ in the unknown $\tilde{y}$ possesses at least the solution y^*, and all solutions $\tilde{y}$ are in $\mathcal{Y}$. In particular, Hoffman's lemma (see, e.g., (Wright, 1997, Lemma A.3)), implies that there exists a solution $\tilde{y} \in \mathcal{Y}$ such that $y - \tilde{y} = O(\|\nabla_x L(x^*, y)\|)$. Thus,

$$\begin{aligned}
\|y - P(y)\| &\le \|y - \tilde{y}\| \\
&= O(\|\nabla_x L(x^*, y)\|) \\
&= O(\|\nabla_x L(x, y)\|) + O(\|\nabla_x L(x, y) - \nabla_x L(x^*, y)\|) \\
&= O(\delta) + O(\|x - x^*\|) \\
&= O(\delta),
\end{aligned}$$

where we used the first part of the proof. Finally, we have $\|x - x^*\| = O(\delta)$ and $\|y - P(y)\| = O(\delta)$, which concludes the proof. □

Lemmas 2 and 3 combine with Assumption 5 to yield the following corollary.

Corollary 2. *Suppose that Assumptions 5 to 7 and 9 hold. Then, for all sufficiently large k,* $\text{dist}((x_k, y_k), \mathcal{S}) = \Theta(\delta_k)$.

Let $\bar{m}$ be the rank of $J(x^*)^T$, with $0 \leq \bar{m} \leq m$. The singular value decomposition of $J(x^*)^T$ may be written as

$$J(x^*)^T = \begin{bmatrix} U_1 & U_2 \end{bmatrix} \begin{bmatrix} \Sigma & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} V_1^T \\ V_2^T \end{bmatrix}, \tag{5.34}$$

where Σ is a diagonal matrix containing the $\bar{m}$ nonzero singular values, U_1 is $n \times \bar{m}$, U_2 is $n \times (n - \bar{m})$, V_1 is $m \times \bar{m}$ and V_2 is $m \times (m - \bar{m})$. Note that $\begin{bmatrix} U_1 & U_2 \end{bmatrix}$ and $\begin{bmatrix} V_1 & V_2 \end{bmatrix}$ are orthogonal and that the columns of U_2 constitute an orthonormal basis for the nullspace of $J(x^*)$. The next result follows (Wright, 1998, Theorem 3.2).

Theorem 11. *Suppose that Assumptions 5 to 9 hold. Suppose that the approximate solution $(\Delta x, -\Delta y)$ of (5.13) satisfies Termination Test 3. Then, for all sufficiently large k,*

$$\Delta x = O(\delta_k) \quad \textit{and} \quad \Delta y = O(\delta_k^{\beta_2}).$$

Proof. If we decompose $\Delta x = U_1 \tilde{x}_{U_1} + U_2 \tilde{x}_{U_2}$ and $\Delta y = V_1 \tilde{y}_{V_1} + V_2 \tilde{y}_{V_2}$, we may rewrite (5.13) as

$$\begin{bmatrix} U_1^T H_k U_1 & U_1^T H_k U_2 & U_1^T J_k^T V_1 & U_1^T J_k^T V_2 \\ U_2^T H_k U_1 & U_2^T H_k U_2 & U_2^T J_k^T V_1 & U_2^T J_k^T V_2 \\ V_1^T J_k U_1 & V_1^T J_k U_2 & -\delta_k I & 0 \\ V_2^T J_k U_1 & V_2^T J_k U_2 & 0 & -\delta_k I \end{bmatrix} \begin{bmatrix} \tilde{x}_{U_1} \\ \tilde{x}_{U_2} \\ -\tilde{y}_{V_1} \\ -\tilde{y}_{V_2} \end{bmatrix} = - \begin{bmatrix} s_{U_1} \\ s_{U_2} \\ s_{V_1} \\ s_{V_2} \end{bmatrix}, \tag{5.35}$$

where

$$\begin{bmatrix} s_{U_1} \\ s_{U_2} \\ s_{V_1} \\ s_{V_2} \end{bmatrix} = \begin{bmatrix} U_1^T (g_k - J_k^T y_k) \\ U_2^T (g_K - J_k^T y_k) \\ V_1^T (c_k - r_k) \\ V_2^T (c_k - r_k), \end{bmatrix}$$

and r_k satisfies (5.31). According to Assumption 6 and Lemma 3, $J_k - J(x^*) = O(\|x_k - x^*\|) = O(\delta_k)$, so that (5.34) yields $U_1^T J_k^T V_1 = \Sigma + O(\delta_k)$, $U_1^T J_k^T V_2 = O(\delta_k)$, $U_2^T J_k^T V_1 = O(\delta_k)$, and

$U_2^T J_k^T V_2 = O(\delta_k)$. We substitute those estimates into (5.35) and obtain

$$\begin{bmatrix} U_1^T H_k U_1 & U_1^T H_k U_2 & \Sigma + O(\delta_k) & O(\delta_k) \\ U_2^T H_k U_1 & U_2^T H_k U_2 & O(\delta_k) & O(\delta_k) \\ \Sigma + O(\delta_k) & O(\delta_k) & -\delta_k I & 0 \\ O(\delta_k) & O(\delta_k) & 0 & -\delta_k I \end{bmatrix} \begin{bmatrix} \tilde{x}_{U_1} \\ \tilde{x}_{U_2} \\ -\tilde{y}_{V_1} \\ -\tilde{y}_{V_2} \end{bmatrix} = - \begin{bmatrix} s_{U_1} \\ s_{U_2} \\ s_{V_1} \\ s_{V_2} \end{bmatrix}.$$

After eliminating $\tilde{y}_{V_2} = -\delta_k^{-1} s_{V_2} + O(\|\tilde{x}_{U_1}\|) + O(\|\tilde{x}_{U_2}\|)$, there remains

$$(M_k + O(\delta_k)) \begin{bmatrix} -\tilde{y}_{V_1} \\ \tilde{x}_{U_2} \\ \tilde{x}_{U_1} \end{bmatrix} = - \begin{bmatrix} s_{U_1} + O(\|s_{V_2}\|) \\ s_{U_2} + O(\|s_{V_2}\|) \\ s_{V_1} \end{bmatrix}, \tag{5.36}$$

where

$$M_k := \begin{bmatrix} \Sigma & U_1^T H_k U_2 & U_1^T H_k U_1 \\ 0 & U_2^T H_k U_2 & U_2^T H_k U_1 \\ 0 & 0 & \Sigma \end{bmatrix}.$$

Assumption 8 and the orthogonality of U ensure that the blocks involving H_k are bounded above by κ. Thus, M_k is uniformly bounded. In addition, M_k is uniformly nonsingular because Σ is nonsingular and Assumption 9 ensures that $U_2^T H_k U_2$ is uniformly positive definite for all sufficiently large k. Thus for all sufficiently large k, $M_k + O(\delta_k)$ is also uniformly nonsingular. Then according to (5.36)

$$\|(\tilde{y}_{V_1}, \tilde{x}_{U_2}, \tilde{x}_{U_1})\| = O(\|(s_{U_1}, s_{U_2}, s_{V_1}, s_{V_2})\|),$$

and

$$\|\tilde{y}_{V_2}\| = O(\delta_k^{-1})\|s_{V_2}\| + O(\|(s_{U_1}, s_{U_2}, s_{V_1}, s_{V_2})\|).$$

From the right-hand side of (5.36), we have

$$\|(s_{U_1}, s_{U_2})\| = \|g_k - J_k^T y_k\| = \|\nabla_x L(x_k, y_k)\| = O(\delta_k),$$

and Termination Test 3 implies that

$$\begin{aligned} \|s_{V_1}\| = \|V_1^T (c_k - r_k)\| &\leq \|c_k - r_k\| \leq \|c(x_k) - c(x^*)\| + \|r_k\| \\ &= O(\|x_k - x^*\|) + O(\delta_k^{1+\beta_2}) = O(\delta_k). \end{aligned}$$

In addition,

$$s_{V_2} = V_2^T(c_k - r_k) = V_2^T \left(c(x^*) + J(x^*)(x_k - x^*) + O(\|x_k - x^*\|^2) - r_k \right).$$

Because $c(x^*) = 0$ and $V_2^T J(x^*) = 0$, there remains

$$\|s_{V_2}\| = O(\|x_k - x^*\|^2) + \|r_k\| = O(\delta_k^2) + O(\delta_k^{1+\beta_2}) = O(\delta_k^{1+\beta_2}).$$

Therefore, $\tilde{y}_{V_2} = O(\delta_k^{\beta_2})$. We have established that $\Delta x = U_1 \tilde{x}_{U_1} + U_2 \tilde{x}_{U_2} = O(\delta_k)$ and $\Delta y = V_1 \tilde{y}_{V_1} + V_2 \tilde{y}_{V_2} = O(\delta_k^{\beta_2})$. □

Note that Theorem 11 holds even if $\beta_2 = 0$ in Termination Test 3. However, in order to establish superlinear convergence, we need to be more demanding on the accuracy of the step computation.

Theorem 12. *Suppose that Assumptions 5 to 10 hold. Suppose that the approximate solution* $(\Delta x, -\Delta y)$ *of* (5.9) *satisfies Termination Test 3 with* $\beta_2 > 0$*. Then, for all sufficiently large* k*,*

$$\delta_k^+ := \delta(x_k^+, y_k^+) = \delta(x_k + \Delta x, y_k + \Delta y) = O(\delta_k^{1+\beta}).$$

where $0 < \beta = \min(\beta_1, \beta_2) \leq 1$*.*

Proof. A straightforward Taylor expansion and the linearity of $\nabla_x L(x, \cdot)$ yield, for all sufficiently large k,

$$\begin{aligned}
\nabla_x L(x_k + \Delta x, y_k + \Delta y) &= \nabla_x L(x_k, y_k) + \nabla_{xx}^2 L(x_k, y_k)\Delta x - J(x_k)^T \Delta y + O(\|\Delta x\|^2) \\
&= \left(\nabla_{xx}^2 L(x_k, y_k) - H_k \right) \Delta x \\
&\quad + H_k \Delta x + \nabla_x L(x_k, y_k) - J(x_k)^T \Delta y + O(\|\Delta x\|^2) \\
&= \left(\nabla_{xx}^2 L(x^*, y^*) - H_k \right) \Delta x + O(\delta_k^2) \\
&= O(\delta_k^{1+\beta_1}),
\end{aligned}$$

where we used the first block equation of (5.13), Assumption 10 and Theorem 11.

Similarly, for all sufficiently large k,

$$\begin{aligned} c(x_k + \Delta x) &= c(x_k) + J(x_k)\Delta x + O(\|\Delta x\|^2) \\ &= r_k - \delta_k \Delta y + O(\|\Delta x\|^2) \\ &= O(\delta_k^{1+\beta_2}) + O(\delta_k^2) \\ &= O(\delta_k^{1+\beta_2}), \end{aligned}$$

where we used the second block equation of (5.13), Termination Test 3 and Theorem 11. The result holds with $\beta := \min(\beta_1, \beta_2) > 0$. □

The following corollary states that, asymptotically, no inner iterations are performed and thus only the extrapolation step of Algorithm 5.3.1 is employed.

Corollary 3. *Suppose that Assumptions 5 to 10 hold. Suppose that the approximate solution* $(\Delta x, -\Delta y)$ *of* (5.13) *satisfies Termination Test 3 with* $\beta_2 > 0$. *Assume that the sequence* $\{\epsilon_k\}$ *is chosen such that*

$$\epsilon_k = \omega(\delta_k^{1+\beta}),$$

where β *is as in Theorem 12. For sufficiently large* k, *the iterates computed at Step 4 of Algorithm 5.3.1 satisfy* $w_{k+1} = w_k^+$ *and* $\delta_k = \|F(w_k)\|$ *converges to zero at the rate* $1 + \beta$.

Proof. The result follows directly from Theorem 12 and the assumption on ϵ_k. □

Because β_1 may be unknown, it is safe to set $\epsilon_k = \Theta(\delta_k)$ in Corollary 3. A consequence of Lemma 2 and Corollary 3 is that $\{w_k\} \to w^*$ R-superlinearly. The next result establishes Q-superlinear convergence to the set $\mathcal{S}$, which, though weaker than convergence to w^*, results from the fact that $\mathcal{Y}$ may be an unbounded set.

Theorem 13. *Under the assumptions of Corollary 3, the sequence* $\{\text{dist}(w_k, \mathcal{S})\}$, *where* $\{w_k\}$ *is generated by Algorithm 5.3.1, converges Q-superlinearly to zero with rate* $1 + \beta$.

Proof. The result follows directly from Corollary 2 and Theorem 12. □

A more precise result follows when (5.9) is solved sufficiently accurately in the sense that $\beta_2 = 1$ in Termination Test 3. The next corollary follows (Wright, 1998, Corollary 4.2).

Corollary 4. *Under the assumptions of Corollary 3 with* $\beta_2 = 1$, *the sequence* $\{w_k\}$ *generated by Algorithm 5.3.1 converges Q-superlinearly to* w^* *with rate* $1 + \beta$, *and* $\|w_k - w^*\| = \Theta(\delta_k)$.

Proof. By Theorem 11, there exists a constant $C > 0$ such that $\|(\Delta x_j, \Delta y_j)\| \leq C\delta_j$ for all sufficiently large j. By Theorem 12, $\{\delta_k\} \to 0$ superlinearly, and thus for all $\ell > k$ sufficiently large, we have

$$\|w_k - w_\ell\| \leq \sum_{j=k}^{\ell-1} \|(\Delta x_j, \Delta y_j)\| \leq C \sum_{j=k}^{\ell-1} \delta_j \leq 2C\delta_k.$$

In the limit, we obtain $\|w_k - w^*\| = O(\delta_k)$. Conversely, Lemma 2 yields $\delta_k = O(\text{dist}(w_k, \mathcal{S})) = O(\|w_k - w^*\|)$. □

We close this section by noting that if Hessian approximations are sufficiently accurate in the sense that $\beta_1 = 1$ in Assumption 10 and if (5.9) is solved sufficiently accurately in the sense that $\beta_2 = 1$ in Termination Test 3, Theorem 12 reveals that $\beta = 1$ and quadratic convergence takes place. In particular, such situation occurs if exact second derivatives are used and (5.9) is solved exactly, e.g., by way of a stable factorization.

It is possible to weaken Assumption 10 and only require $(H_k - \nabla^2_{xx} L(x^*, y^*))\Delta x = o(\|\Delta x\|)$, which is closer to the original Dennis and Moré (1977) condition. In that case, the conclusion of Theorem 12 changes to $\delta_k^+ = o(\delta_k)$. Corollary 3, Theorem 13 and Corollary 4 all remain valid except that the rate of superlinear convergence cannot be specified.

Our requirements on the quality of the Hessian approximation and on the accuracy of the step computation are substantially weaker than those of, e.g., Armand and Omheni (2015), who require exact steps and the stringent bound $H_k - \nabla_{xx} L(x_k, y_k) = O(\delta_k)$. In view of Theorem 11, the latter is akin to requiring exact second derivatives.

5.6 Implementation and Numerical Results

In the following section we examine the practical behavior of Algorithms 5.3.1 and 5.3.2 and specify the details of our implementation. Our implementation is written in Python with the help of the open-source package NLP.py (Arreckx et al., 2016), a programming environment for designing numerical optimization methods. A Python implementation of LSMR is available in the PyKrylov package (Orban, 2009), a library of Krylov methods in pure Python.

Initial Lagrange multipliers Given a user-defined starting point x_s, the vector of Lagrange multipliers y_s is obtained as least-square solutions of $\nabla L(x_s, y) = 0$, i.e., by solving the linear system

$$\begin{bmatrix} I & J_s^T \\ J_s & -\zeta I \end{bmatrix} \begin{bmatrix} v \\ -y \end{bmatrix} = - \begin{bmatrix} g_s \\ 0 \end{bmatrix},$$

using MA57 (Duff, 2004) and discarding v or, alternatively, using LSMR to solve

$$\min_{y \in \mathbb{R}^m} \ \tfrac{1}{2}\|J_s^T y - g_s\|^2 + \tfrac{1}{2}\zeta\|y\|^2.$$

In our implementation, ζ is set to 10^{-8}.

Initial point Using the starting point x_s and the least-squares vector of Lagrange multipliers y_s, we compute $(\Delta x_s, \Delta y_s)$ by solving (5.13) with $\delta = 0$. If $\|F(x_s + \Delta x_s, y_s + \Delta y_s)\| < \|F(x_s, y_s)\|$, we select the improved starting point $(x_0, y_0) := (x_s + \Delta x_s, y_s + \Delta y_s)$. Otherwise (x_0, y_0) is set to (x_s, y_s).

Penalty parameter The initial penalty parameter is set to $\delta_0 = \min\{0.1, \|F(w_0)\|\}$. In Step 3 of Algorithm 5.3.1, δ_k is chosen as

$$\delta_k = \max\left\{\min\{\|F(w_k)\|,\ 0.9\delta_{k-1},\ \delta_{k-1}^{1.1}\},\ \delta_{\min}\right\}$$

where $\delta_{\min}$ is a lower bound imposed on the penalty parameter. This rule ensures that δ_k does not deviate much from the value selected by the augmented Lagrangian mechanism in the global regime, and is set to $\|F(w_k)\|$ asymptotically so that Assumption 7 of the local convergence analysis is satisfied.

When w_k^+ does not satisfy (5.14), a sequence of inner iterations using Algorithm 5.3.2 is started from the initial guess $w_{k,0} = w_k$.

Even though global convergence of Algorithm 5.3.2 is promoted by way of a Wolfe linesearch, we have found that a simple Armijo linesearch is nearly as effective. In the first Wolfe condition, we use $c_1 = 10^{-4}$. At the end of the inner iterations, the current value of the penalty parameter is returned to the outer iteration and is used as δ_k.

Optimality conditions Optimality is declared when the norm of the optimality conditions at w_k satisfies

$$\|F(w_k)\| < \epsilon_{\text{tol}}\ \|F(w_0)\|$$

where $\epsilon_{\text{tol}} = 10^{-6}$. In (5.15), we set $\epsilon_k = 10\delta_k$ and $\theta = 0.99$.

Exact system solves We use the linear solver MA57 to solve (5.13) and (5.16). The primal regularization parameter ρ_k is updated until a correct inertia is detected according to (Wächter and Biegler, 2006, Algorithm IC), with the same values for the various constants.

Inexact system solves In Step 3 of Algorithm 5.3.1 and Step 3 of Algorithm 5.3.2, LSMR is used to solve the preconditioned linear least-squares problem (5.20). The preconditioner, $H_{k,j}^{-1}$, is obtained by maintaining a limited-memory BFGS approximation of the Hessian of the Lagrangian in inverse form. It is well known that a standard BFGS approximation is ineffective in the presence of constraints because the Hessian of the Lagrangian is typically indefinite at a solution—see, e.g., (Byrd et al., 1992). If $s_k = x_{k+1} - x_k$ and $t_k = \nabla L(x_{k+1}, y_{k+1}) - \nabla L(x_k, y_{k+1})$, the curvature condition $s_k^T t_k > 0$ is unlikely to hold asymptotically, and the pair (s_k, t_k) will be rejected.

Powell (1978) suggests to use a damped BFGS update that compensates for the lack of positive definiteness in the Hessian at the solution. His approach is based on the Hessian of the Lagrangian, instead of its inverse. In the current work, we follow the same idea, but formulated in terms of $B_k = H_k^{-1}$. Let $q_k := \theta_k s_k + (1 - \theta_k) B_k t_k$ where $\theta_k \in (0, 1]$ is defined as

$$\theta_k = \begin{cases} 1 & \text{if } s_k^T t_k \geq \eta\, t_k^T B_k t_k \\ (1-\eta)\, t_k^T B_k t_k / (t_k^T B_k t_k - s_k^T t_k) & \text{otherwise,} \end{cases}$$

for some $\eta \in (0, 1)$. In our implementation, we set $\eta := 0.2$. A straightforward derivation similar to that of Powell (1978) shows that if B_k is positive definite, B_{k+1} is also positive definite. Thus starting this damped BFGS approximation by a scaled identity matrix $B_0 = \gamma_0 I$ with $\gamma_0 > 0$ ensures positive definiteness of all subsequent approximations. In addition, only a small number of pairs (s_k, q_k) is stored so as to provide a limited-memory version of the damped BFGS method. Application of this approximation to a vector is performed using the two loop recursion.

Another way of ensuring the positive-definiteness of H_k is to maintain a BFGS approximation of the Hessian of the augmented Lagrangian ϕ (Byrd et al., 1992). This proposal is motivated by the fact that if δ is sufficiently large, the Hessian of $\phi(x, y; \rho, \delta)$ is positive definite for all (x, y) close to an isolated solution. However, in our numerical experiments, results were not as good as with Powell's damped BFGS. A reason for this is that when δ increases, convergence of the BFGS approximation is disrupted. Similar observations were also reported by Gill and Wong (2011).

In Termination Test 3, we set $\beta_2 = 0.5$ and $\mu = 0.2$. During the inner iterations, $\gamma_j = 10^{-4}$ for all j in Termination Test 1.

5.6.1 Numerical Experiments

We perform preliminary comparative tests between our implementation of Algorithm 5.3.1, named RegSQP, AUGLAG (Arreckx et al., 2015) and IPOPT 3.12.1 (Wächter and Biegler, 2006) on a few problems from the CUTEst (Gould et al., 2015) and COPS (Dolan et al., 2004) collections. All models are formulated using the AMPL modeling language (Fourer et al., 2003). Problems tested range from 2 to 20,192 variables and from 1 to 10,000 equality constraints, most of them having more than 1,400 variables and 900 constraints. Table 5.1 summarizes the characteristics of selected test problems: number of variables, constraints, and nonzero elements in the Jacobian and Hessian. The problems of Table 5.1 were chosen because the default version of IPOPT performs well on them and the parameter values and updates described above show encouraging performance, both in the direct and iterative variants. We are currently trying to identify parameter values and updates that perform well on a larger test set.

IPOPT is a good comparison when using exact second derivatives because in the absence of inequality constraints, it reduces to a filter/linesearch factorization-based SQP method that is similar to RegSQP. An important difference is that IPOPT treats the parameter δ_k as a static ad-hoc constraint regularization parameter in case rank deficiency of the Jacobian is detected. Because IPOPT also offers the possibility to use L-BFGS approximations of the second derivatives, we use it as a basis for comparison although, strictly speaking, it is not factorization free. It does however allow us to compare RegSQP against a typical SQP implementation that uses quasi-Newton Hessian approximations. An additional difference is that IPOPT sets H_k to a (undamped) L-BFGS approximation of the Hessian of the Lagrangian, even though the latter cannot be expected to be positive definite in the limit. When using L-BFGS approximations, a better candidate for comparison is AUGLAG, an ℓ_∞-norm trust-region augmented Lagrangian method similar in spirit to LANCELOT (Conn et al., 1992), with the difference that trust-region subproblems are solved using our matrix-free implementation of TRON (Lin and Moré, 1998). The main differences with the original implementation of TRON are that the Hessian is only required as an operator, and that the conjugate gradient method used to minimize the model on a face of the trust region does not use a preconditioner. AUGLAG is thus completely factorization free.

We do not apply any scaling procedure to the problems. The IPOPT option `nlp_scaling_method` is set to `none`. We also set the IPOPT options `tol`, `dual_inf_tol` and `constr_viol_tol` to 10^{-6}. IPOPT stops as soon as

$$\max\left\{\frac{\|\nabla_x L(x,y)\|_\infty}{s_d},\ \|c(x)\|_\infty\right\} \leq 10^{-6},$$

Table 5.1 Characteristics of selected test problems from CUTEst and COPS: number of variables (nvar), constraints (ncon), nonzero elements in the Jacobian (nnz(J)) and in the Hessian (nnz(H))

name	nvar	ncon	nnz(J)	nnz(H)
elec-1	150	50	150	11325
elec-2	300	100	300	45150
elec-3	600	200	600	180300
aug2d	20192	9996	39984	19800
aug3d	3873	1000	6546	2673
aug3dc	3873	1000	6546	3873
bt1	2	1	2	2
dtoc1l	14985	9990	82889	14985
dtoc1na	1485	990	15735	6385
dtoc1nb	1485	990	15735	6385
dtoc1nc	1485	990	15735	6385
eigencco	30	15	125	465
gridnetb	13284	6724	26568	13284
hager1	10000	5000	14999	5001
hager2	10000	5000	14999	14999
hager3	10000	5000	14999	24998
integreq	100	100	10000	100

where $s_d > 0$ is a scaling factor. AUGLAG is set to stop when

$$\|\nabla_x L(x, y))\|_\infty \leq 10^{-6}\|\nabla_x L(x_0, y_0))\|_\infty \quad \text{and} \quad \|c(x)\|_\infty \leq 10^{-6}\|c(x_0)\|_\infty.$$

The maximum number of iterations is set to 3000 for all solvers and a limit of 1 hour of run time is imposed. Tests are conducted on a MacBook Pro 2.4 GHz equipped with a Intel Core i5 processor and 8 Gb of memory running OSX 10.10.5.

As a measure of efficiency of all the algorithms, we use the number of function and gradient calls and either the number of Hessian calls when exact second derivatives are used, or the number of Jacobian vector products when L-BFGS approximations are used. We deliberately choose not to include cpu time comparisons because our algorithm and AUGLAG are implemented using a high-level language (Python) whereas IPOPT is implemented in a compiled language (C++).

We first compare both methods using exact second derivatives. Both IPOPT and RegSQP use MA57 (Duff, 2004). Table 5.2 shows a comparison in terms of number of objective, gradient and Hessian evaluations. The results show that IPOPT and RegSQP perform similarly on those problems. Note however that RegSQP seems less efficient than IPOPT in terms of function calls and number of Hessian evaluations. These observations are encouraging because IPOPT is a mature implementation that has benefited from years of development.

Next, we compare both methods using L-BFGS approximations. In this case, IPOPT does not use an iterative method to solve augmented systems, but relies on a factorization and on the Sherman-Morrison-Woodbury formula to take the low-rank update into account. All solvers store 6 pairs in the history. Table 5.3 presents the results of the comparison. As IPOPT does not provide the number of Jacobian-vector products, we estimate it as $\#J$ calls $\times \min\{m, n\}$. RegSQP significantly outperforms IPOPT in terms of number of Jacobian-vector products. We also note that AUGLAG requires fewer Jacobian-vector products than RegSQP but generally requires substantially more function and gradient evaluations.

The analysis of Sections 5.4 and 5.5 does not rely on the LICQ. The following two small examples, `HS026` and `HS039`, demonstrate the behavior of our algorithm on problems that do not satisfy the LICQ. One way to create a degenerate problem is to add an additional constraint of the form $c_1(x) = c_1(x)^2$ to the model, where c_1 is the first constraint of the model. The Jacobian is thus rank deficient everywhere.

IPOPT regularizes the augmented matrix by adding a nonzero diagonal term $-\delta_c I$ to the $(2, 2)$ block when the Jacobian appears to be rank deficient. The parameter δ_c is controlled by the `jacobian_regularization_value` option whose default value is 10^{-8}.

Table 5.2 Comparison of IPOPT and RegSQP using exact second derivatives in terms of number of function calls (#f), gradient calls (#g) and Hessian calls (#H))

	IPOPT			RegSQP		
	#f	#g	#H	#f	#g	#H
elec-1	47	42	41	123	76	45
elec-2	262	187	186	304	162	125
elec-3	420	293	292	290	173	115
aug2d	2	2	1	2	2	1
aug3d	3	3	2	2	2	1
aug3dc	2	2	1	2	2	1
bt1	11	8	7	12	10	8
dtoc1l	7	7	6	9	7	6
dtoc1na	7	7	6	10	8	7
dtoc1nb	7	7	6	7	7	6
dtoc1nc	21	16	15	22	15	14
eigencco	13	13	12	21	17	14
gridnetb	2	2	1	2	2	1
hager1	2	2	1	2	2	1
hager2	2	2	1	2	2	1
hager3	2	2	1	2	2	1
integreq	4	4	3	4	4	3

Table 5.3 Comparison of AUGLAG, IPOPT and RegSQP using L-BFGS approximations in terms of number of function calls (#f), gradient calls (#g) and products with the Jacobian or its transpose (#Jprod))

	AUGLAG			IPOPT			RegSQP		
	#f	#g	#Jprod	#f	#g	#Jprod	#f	#g	#Jprod
elec-1	6862	5154	5154	306	177	8850	375	197	3423
elec-2	1069	766	766	746	434	43400	548	255	4799
elec-3	1783	1241	1241	1832	1055	211000	2672	728	9541
aug2d	11224	8357	8357	2	2	19992	51	36	16243
aug3d	2190	1857	1857	3	3	3000	37	26	1982
aug3dc	488	250	250	2	2	2000	17	13	727
bt1	224	67	67	9	7	14	54	11	69
dtoc1l	938	657	657	129	26	259740	33	24	3123
dtoc1na	801	540	540	24	22	21780	33	23	2883
dtoc1nb	1000	683	683	23	23	22770	43	28	3319
dtoc1nc	1322	902	902	54	44	43560	104	81	7485
eigencco	821	604	604	98	94	1410	139	72	1393
gridnetb	52094	31477	31477	425	103	692572	129	81	50239
hager1	6045	5312	5312	4	4	20000	31	26	7577
hager2	7103	4117	4117	6	6	30000	47	28	7330
hager3	10834	9895	9895	6	6	30000	82	34	14491
integreq	89	64	64	4	4	400	12	10	153

For these two problems, exact second derivatives and the direct solver MA57 were used. Sections 5.6.1 and 5.6.1 show a comparison between IPOPT and RegSQP in terms of number of calls to the objective, gradient and Hessian when both use exact second derivatives. Only two examples are shown here but they are representative of the behavior of the two solvers on degenerate problems. Although IPOPT and RegSQP behave similarly on nondegenerate problems, the situation is different on degenerate problems. The performance of RegSQP does not degrade too much in the presence of degeneracy, while degeneracy noticeably impairs IPOPT's performance.

Table 5.4 HS026: $n = 3$, $m = 1$

	IPOPT		RegSQP	
	original	degenerate	original	degenerate
# objective evals	26	267	17	54
# gradient evals	26	55	18	40
# Hessian evals	25	54	17	39

Table 5.5 HS039: $n = 4$, $m = 2$

	IPOPT		RegSQP	
	original	degenerate	original	degenerate
# objective evals	14	114	12	17
# gradient evals	14	51	13	18
# Hessian evals	13	50	12	17

5.7 Discussion

The main contribution of this paper is the formulation and analysis of an algorithm for equality-constrained optimization that combines the favorable global properties of augmented Lagrangian methods and local properties of stabilized SQP methods. The use of positive-definite limited-memory approximations to the Hessian of the Lagrangian presents the significant advantage that the linear system encountered at each iteration is always SQD. An appropriate interpretation of that system in terms of a linear least-squares problem permits efficient inexact system solves and an entirely factorization-free implementation. The numerical results of §5.6 indicate that the proposed method is robust and efficient, even when the problem is degenerate.

The use of our BFGS-LSMR strategy for solving (5.13) or (5.16) inexactly is not restricted to the specific scope of the present research. It could also be employed when applying a

quadratic penalty method to (5.1) in the same vein as described by Armand et al. (2014). Following the procedure of Section 5.2 leads to the fully regularized subproblem

$$\min_{x\in\mathbb{R}^n, r\in\mathbb{R}^m} \; f(x) + \tfrac{1}{2}\rho_k\|x - x_k\|^2 + \tfrac{1}{2}\delta_k\|r\|^2 \quad \text{subject to } \; c(x) + \delta_k r = 0, \tag{5.37}$$

for $\delta_k > 0$, $\rho_k \geq 0$ and for some new variables r. Applying Newton's method to the KKT conditions of (5.37) yields

$$\begin{bmatrix} H_k + \rho_k I & J_k^T \\ J_k & -\delta_k I \end{bmatrix} \begin{bmatrix} \Delta x \\ -\Delta y \end{bmatrix} = - \begin{bmatrix} g_k - J_k^T y_k \\ c_k + \delta_k y_k \end{bmatrix}. \tag{5.38}$$

Note that the coefficient of (5.38) is the same as that of (5.16), only the right-hand side differs.

The local convergence properties of Section 5.5 assume that quasi-Newton approximations converge superlinearly to the exact Hessian at the solution along the primal steps. Byrd et al. (1992) establish that fast local convergence of a SQP method can take place provided H_k is defined as the BFGS approximation of the Hessian of the *augmented* Lagrangian. It may be possible to transpose their results to the present context because (5.9) is precisely a step on an augmented Lagrangian. Unfortunately, the performance of the augmented Lagrangian approximation is poor compared to that of the damped BFGS update of Powell (1978). Local convergence properties of the damped update remains an open question. Powell proves that if convergence occurs, it does so at a R-superlinear rate. Further exploration of those considerations is beyond the scope of the present paper and is left for future research.

Our method can be extended to problems with inequality constraints. A first approach could be to use an augmented Lagrangian function that takes inequalities into account, such as that of Birgin and Martínez (2014). Another approach is to reformulate the optimization problem by adding slack variables and treat bounds via a logarithmic barrier. The precise form of the Newton equations used is likely to be of crucial importance in practice when designing the factorization-free variant (Greif et al., 2014).

Finally, allowing the penalty parameter to increase during the inner iterations as proposed in Armand and Omheni (2015) might further improve performance.

REFERENCES

M. Arioli and D. Orban (2013). Iterative methods for symmetric quasi-definite linear systems—part I: Theory. Cahier du GERAD G-2013-32, GERAD, Montréal, QC, Canada.

P. Armand, J. Benoist, R. Omheni, and V. Pateloup (2014). Study of a primal-dual algorithm for equality constrained minimization. *Computational Optimization and Applications*, 59(3):405–433. doi: 10.1007/s10589-014-9679-3.

P. Armand, J. Benoist, and D. Orban (2012). From global to local convergence of interior methods for nonlinear optimization. *Optimization Methods and Software*, 28(5):1051–1080. doi: 10.1080/10556788.2012.668905.

P. Armand and R. Omheni (2015). A globally and quadratically convergent primal-dual augmented Lagrangian algorithm for equality constrained optimization. *Optimization Methods and Software*. doi: 10.1080/10556788.2015.1025401. Online First.

S. Arreckx, A. Lambe, J. R. R. A. Martins, and D. Orban (2015). A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization. *Optimization and Engineering*, pages 1–26. doi: 10.1007/s11081-015-9287-9.

S. Arreckx, D. Orban, and N. van Omme (2016). NLP.py — a large-scale optimization toolkit in Python. Cahier du GERAD G-2016-42, GERAD, Montréal, QC, Canada.

D. P. Bertsekas (1996). *Constrained optimization and Lagrange multiplier methods*. Academic press. doi: 10.1016/b978-0-12-093480-5.50005-2.

E. G. Birgin and J. M. Martínez (2014). *Practical augmented Lagrangian methods for constrained optimization*. SIAM. doi: 10.1137/1.9781611973365.

P. T. Boggs and J. W. Tolle (1995). Sequential quadratic programming. *Acta Numerica*, 4:1–51. doi: 10.1017/S0962492900002518.

R. H. Byrd, F. E. Curtis, and J. Nocedal (2009). An inexact Newton method for nonconvex equality constrained optimization. *Mathematical Programming*, 122(2):273–299. doi: 10.1007/s10107-008-0248-3.

R. H. Byrd, J. Nocedal, and R. B. Schnabel (1994). Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1):129–156. doi: 10.1007/BF01582063.

R. H. Byrd, R. A. Tapia, and Y. Zhang (1992). An SQP augmented lagrangian BFGS algorithm for constrained optimization. *SIAM Journal on Optimization*, 2(2):210–241.

A. R. Conn, N. I. M. Gould, and P. L. Toint (1992). LANCELOT*: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer Publishing Company, Incorporated, first edition.

J. E. Dennis and J. J. Moré (1977). Quasi-Newton methods, motivation and theory. *SIAM Review*, 19(1):46–89. doi: 10.1137/1019005.

J. E. J. Dennis and R. B. Schnabel (1996). *Numerical methods for unconstrained optimization and nonlinear equations*, volume 16. SIAM. doi: 10.1137/1.9781611971200.

E. D. Dolan, J. J. Moré, and T. S. Munson (2004). Benchmarking Optimization Software with COPS 3.0. Technical Report ANL/MCS-TM-273, Argonne National Laboratory, Mathematics and Computer Science division.

I. S. Duff (2004). MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144. doi: 10.1145/992200.992202.

D. Fernández, E. A. Pilotta, and G. A. Torres (2012). An inexact restoration strategy for the globalization of the sSQP method. *Computational Optimization and Applications*, 54(3):595–617. doi: 10.1007/s10589-012-9502-y.

D. Fernàndez and M. Solodov (2010). Stabilized sequential quadratic programming for optimization and a stabilized Newton-type method for variational problems. *Mathematical Programming*, 125(1):47–73. doi: 10.1007/s10107-008-0255-4.

D. C.-L. Fong and M. A. Saunders (2011). LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971. doi: 10.1137/10079687X.

R. Fourer, D. M. Gay, and B. W. Kernighan (2003). *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press / Brooks/Cole Publishing Company, second edition.

M. P. Friedlander and D. Orban (2012). A primal-dual regularized interior-point method for convex quadratic programs. *Mathematical Programming Computation*, 4(1):71–107. doi: s12532-012-0035-2.

J. Gauvin (1977). A necessary and sufficient regularity condition to have bounded multipliers in nonconvex programming. *Mathematical Programming, Series B*, 12:136–138.

P. Gill and D. Robinson (2013). A globally convergent stabilized SQP method. *SIAM Journal on Optimization*, 23(4):1983–2010. doi: 10.1137/120882913.

P. E. Gill and E. Wong (2011). Sequential quadratic programming methods. In J. Lee and S. Leyffer, editors, *Mixed Integer Nonlinear Programming*, volume 154 of *The IMA Volumes*

in Mathematics and its Applications, pages 147–224. Springer New York. doi: 10.1007/978-1-4614-1927-3_6.

N. I. M. Gould (1985). On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem. *Mathematical Programming*, 32(1):90–99. doi: 10.1007/BF01585660.

N. I. M. Gould, D. Orban, and P. L. Toint (2015). CUTEst: a Constrained and Unconstrained Testing Environment with safe threads. *Computational Optimization and Applications*, 60(3):545–557. doi: 10.1007/s10589-014-9687-3.

C. Greif, E. Moulding, and D. Orban (2014). Bounds on the eigenvalues of matrices arising from interior-point methods. *SIAM Journal on Optimization*, 24(1):49–83. doi: 10.1137/120890600.

W. W. Hager (1999). Stabilized sequential quadratic programming. *Computational Optimization and Applications*, 12(1):253–273. doi: 10.1023/A:1008640419184.

M. Heinkenschloss and D. Ridzal (2014). A matrix-free trust-region sqp method for equality constrained optimization. *SIAM Journal on Optimization*, 24(3):1507–1541.

A. F. Izmailov and M. V. Solodov (2012). Stabilized SQP Revisited. *Mathematical Programming*, 133(1–2):93–120. doi: 10.1007/s10107-010-0413-3.

A. F. Izmailov, M. V. Solodov, and E. I. Uskov (2015). Combining stabilized SQP with the augmented Lagrangian algorithm. *Computational Optimization and Applications*, 62(2):405–429. doi: 10.1007/s10589-015-9744-6.

C.-J. Lin and J. J. Moré (1998). Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9:1100–1127. doi: 10.1137/S1052623498345075.

D. C. Liu and J. Nocedal (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1):503–528. doi: 10.1007/BF01589116.

J. Nocedal and S. J. Wright (2006). *Numerical Optimization*. Springer, New York, second edition. doi: 10.1007/b98874.

D. Orban (2009). PyKrylov: Krylov subspace methods in pure Python. `github.com/PythonOptimizers/pykrylov`.

C. Paige and M. A. Saunders (1975). Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629. doi: 10.1137/0712047.

C. C. Paige and M. A. Saunders (1982). LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71. doi: 10.1145/355984.355989.

M. J. D. Powell (1978). A fast algorithm for nonlinearly constrained optimization calculations. In *Lecture Notes in Mathematics*, pages 144–157. Springer Science + Business Media. doi: 10.1007/bfb0067703.

R. T. Rockafellar (1976). Augmented Lagrangians and applications of the proximal point algorithm in convex programming. *Mathematics of Operations Research*, 1:97–116. doi: 10.1287/moor.1.2.97.

R. J. Vanderbei (1995). Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113. doi: 10.1137/0805005.

A. Wächter and L. T. Biegler (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, 106:25–57. doi: 10.1007/s10107-004-0559-y.

R. B. Wilson (1963). *A Simplicial Method for Convex Programming*. PhD thesis, Harvard University, Boston, USA.

S. J. Wright (1997). *Primal-Dual Interior-Point Methods*. SIAM. doi: 10.1137/1.9781611971453.

S. J. Wright (1998). Superlinear Convergence of a Stabilized SQP Method to a Degenerate Solution. *Computational Optimization and Applications*, 11(3):253–275. doi: 10.1023/A:1018665102534.

S. J. Wright (2005). An algorithm for degenerate nonlinear programming with rapid local convergence. *SIAM Journal on Optimization*, 15(3):673–696. doi: 10.1137/030601235.

CHAPITRE 6 PRÉSENTATION GÉNÉRALE DE NLP.py

Un développeur d'algorithmes d'optimisation qui souhaite implémenter et tester un nouvel algorithme doit souvent le faire à partir de zéro. Le but de ce travail est de faciliter le développement d'algorithmes en fournissant une infrastructure extensible pour mettre en oeuvre de nouveaux algorithmes. Le présent article s'intéresse à la création d'un écosystème qui vise le développement d'algorithmes d'optimisation, appelé NLP.py, pour *Non Linear Progamming in Python* (programmation non linéaire en Python). Cet environnement s'adresse aussi bien aux chercheurs en optimisation qu'aux étudiants désireux de découvrir ou d'approfondir les multiples facettes de l'optimisation. D'une part, il facilite la modélisation de problèmes d'optimisation. Et d'autre part, il simplifie le développement de nouveaux algorithmes d'optimisation dans un langage de haut-niveau aussi puissant que Python.

NLP.py regroupe un grand nombre de blocs essentiels à la construction d'algorithmes d'optimisation tels que des techniques de globalisation, des outils permettant de résoudre des systèmes linéaires denses ou creux, de même que des solveurs complets. Chacun d'eux est détaillé dans le chapitre suivant. Le chercheur peut alors combiner ces blocs afin de créer un nouvel algorithme d'optimisation. L'implémentation d'un algorithme se divise principalement en deux tâches : la modélisation et la conception algorithmique. Dans la première, le chercheur doit définir un modèle dans lequel il choisit d'où proviennent les dérivées, la façon dont sont stockées les matrices ou encore l'utilisation de dérivées secondes approchées. Dans la deuxième, il faut déterminer le type d'algorithme, le choix de la technique de globalisation et enfin la façon de résoudre les systèmes linéaires. La Figure 6.1 illustre les relations entre ces principales tâches.

NLP.py utilise Python comme langage de haut niveau dans lequel les algorithmes d'optimisation peuvent être facilement programmés sans se soucier des détails ne se rapportant pas à l'optimisation.

Pour des raisons d'efficacité, les tâches de calculs intensifs sont mises en œuvre en Cython (Behnel *et al.*, 2011). Celles-ci incluent la résolution de systèmes symétriques définis positifs de même que de systèmes symétriques indéfinis par des méthodes directes (factorisation LU ou QR) ou par des méthodes itératives.

Les algorithmes d'optimisation, quant à eux, consistent en des programmes écrits purement en Python et sont donc courts, faciles à lire et à modifier. De simples appels de fonction donnent alors accès aux routines de bas niveau. Grâce à ce choix de conception, le chercheur est en mesure de se concentrer sur la logique des algorithmes plutôt que sur les subtilités

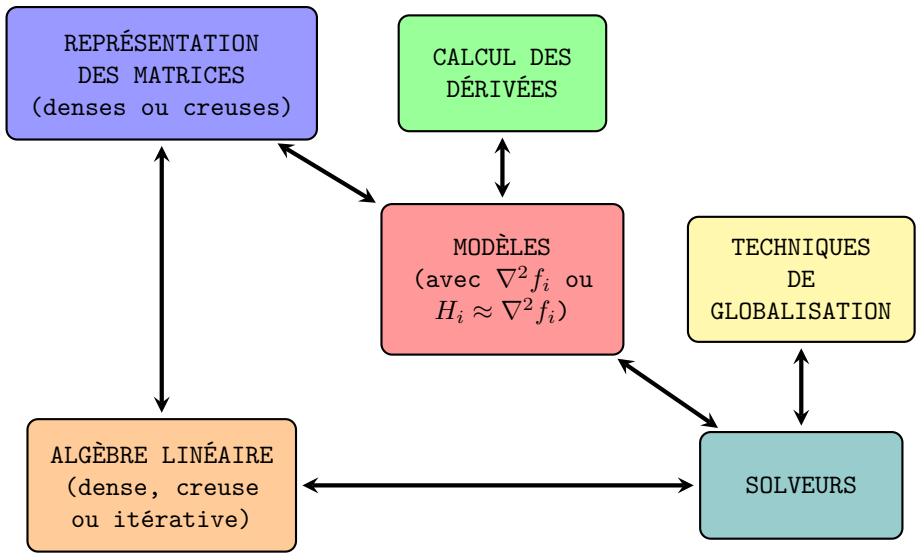

Figure 6.1 Structure de NLP.py.

techniques de son implémentation.

Plusieurs algorithmes d'optimisation ont déjà été implémentés avec succès dans NLP.py. Notamment, des algorithmes permettant de résoudre des problèmes linéaires, des problèmes quadratiques convexes, des problèmes sans contraintes non convexes, de même que des problèmes ne contenant que des bornes ou encore des problèmes plus généraux possédant des contraintes non linéaires. De plus, les algorithmes AUGLAG du Chapitre 4 et RegSQP du Chapitre 5 sont tous deux basés sur les blocs et mécanismes fournis par NLP.py.

NLP.py est un logiciel open-source, disponible gratuitement sur GitHub à l'adresse suivante : `github.com/PythonOptimizers/NLP.py.git`. La documentation ainsi que des exemples d'utilisation y sont également présents. Dans l'organisation GitHub, `PythonOptimizers`, se trouvent aussi tous les packages Cython/Python que nous avons créés et que nous utilisons au sein de NLP.py :

- CySparse : une bibliothèque de matrices creuses écrites en Cython, qui se distingue de PySparse par les nombreux types de données disponibles
- cygenja : un outil de génération de code basé sur Jinja2 qui permet d'imiter le mécanisme des «templates» du C++ mais pour Cython
- PyKrylov : une collection de méthodes de Krylov écrites en pur Python
- qr_mumps.py : une interface Cython/Python à la factorisation QR de Buttari (2013)
- SuiteSparse.py : un ensemble d'interfaces Cython/Python à quelques unes des routines présentes dans le projet SuiteSparse (Davis, 2004, 2011) en particulier UMFPACK, CHOLMOD et SPQR

— MUMPS.py : une interface Cython/Python à la factorisation LU de Amestoy *et al.* (1998)
— HSL.py : un ensemble d'interfaces Cython/Python à certaines routines de la Harwell Subroutine Library (2007)
— LLDL.py : une factorisation LDL^T à mémoire limitée en pur Python basée sur les travaux de Orban (2014).

Ces packages ne font pas partie intégrante de NLP.py parce que nous pensons qu'ils peuvent être utiles dans d'autres contextes que celui de NLP.py.

CHAPITRE 7 ARTICLE 3 : NLP.py: AN OBJECT-ORIENTED ENVIRONMENT FOR LARGE-SCALE OPTIMIZATION

Sylvain Arreckx
École Polytechnique de Montréal & GERAD, Canada
Dominique Orban
École Polytechnique de Montréal & GERAD, Canada
Nikolaj van Omme
École Polytechnique de Montréal & GERAD, Canada

Abstract

NLP.py is a programming environment to model continuous optimization problems and to design computational methods in the high-level and powerful Python language with performance-critical parts implemented in Cython, a low-level superset of Python that compiles to machine code. With the aim of designing numerical methods, NLP.py is accompanied by an extensive set of building blocks to solve the linear algebra and subproblems typically encountered in the solution of large-scale convex and nonconvex problems, including direct and iterative method for linear systems, linesearch strategies, trust-region subproblems, and bound-constrained subproblems. NLP.py supports several sparse matrix packages, including our own novel CySparse library. NLP.py features turnkey algorithms for problems with specific structure along with tools to assess performance. The extensible nature of NLP.py combines with the might and ubiquity of Python to make it a powerful development and analysis environment for optimization researchers and practitioners.

7.1 Introduction

We describe an open source Python programming environment for optimization that combines the advantages of scripting and compiled languages. NLP.py, which stands for *nonlinear programming in Python*, is written with the design of large-scale novel optimization methods in mind but may also be used as a set of optimization solvers. Rather than trying to offer as wide a selection as possible of methods, a conscious decision in the design of NLP.py is to implement a few efficiency-driven cutting-edge methods that represent recent research, and to provide access to effective commonly-used building blocks. With those building blocks, users

Ce chapitre correspond à un article soumis à ACM Transactions on Mathematical Software (TOMS).

are able to assemble existing methods for testing, or to prototype novel methods for research. As a consequence, the researcher is able to concentrate on the logic of the algorithms rather than on the intricacies of core features such as a Wolfe linesearch or the efficient solution of a symmetric indefinite linear system. One of the main design goals is to let users experiment with variants of computational methods by swapping an implementation detail for another, e.g., changing a linesearch scheme for another, or a quasi-Newton approximation for another. The Python language itself reinforces this aspect, being non-intrusive, ubiquitous, and easy to learn.

Core NLP.py focuses on optimization and is accompanied by satellite packages that users can install as needed and focusing mainly on linear algebra operations, including sparse matrix libraries, symmetric indefinite factorizations, sparse QR factorization, and Krylov methods.

NLP.py has already been used successfully to implement numerical algorithms for linear programming and unconstrained, bound-constrained, and nonlinearly-constrained optimization. Those algorithms have been applied to solve, among others, problems from the CUTE/AMPL collection (Gould et al., 2003b; Vanderbei, 2009), the COPS collection (Dolan et al., 2004), the McMPEC collection of problems with complementarity constraints (Leyffer, 2004; Coulibaly and Orban, 2012) and structural optimization problems formulated as programs with vanishing constraints (Curatolo, 2008). More recently, it has been used to develop a factorization-free augmented Lagrangian implementation for structural design problems that arise in aircraft wing design optimization (Arreckx et al., 2015).

It is our contention that NLP.py offers the flexibility, power and ease of use for teaching optimization, studying numerical methods, researching new algorithms and exploring new ideas, all the while retaining efficiency and promoting expandability and code reuse at all levels. NLP.py may be obtained from `github.com/PythonOptimizers/NLP.py.git`.

Related Work

Object-oriented languages such as C++ have gained considerable momentum in the scientific computing community in the past two decades, partly because of the flexibility that they offer and partly because of their popularity in businesses and research institutions. Object-oriented languages offer a degree of abstraction that lets programmers devise natural, powerful and expandable toolkits. Among the advantages offered by object-oriented languages, abstraction, inheritance and polymorphism allow the programmer to specialize a given method of a given object via subclassing and overriding. This is a useful feature in the context of optimization algorithms because it allows to subclass, say, an algorithm, and specialize it.

C++ is often the language of choice for large-scale object-oriented solvers and libraries. Meza et al. (2007) propose the OPT++ object-oriented C++ toolkit for optimization that distinguishes between an algorithm-independent class hierarchy for problems and a class hierarchy for numerical methods that is based on common algorithmic traits. OPT++ supports parallel computing capabilities and simulation-based optimization, which make it a promising library for large-scale real-world problems. OOQP (Gertz and Wright, 2003) is a C++ package for convex quadratic programming that allows the solution of problems with specialized structure via subclassing and polymorphism. The TAO Toolkit for Advanced Optimization of Benson et al. (2016) is an object-oriented framework for optimization that is largely based on the parallel linear algebra capabilities of the PETSc library (Balay et al., 2009). OOPS (Gondzio and Sarkissian, 2003; Gondzio and Grothey, 2007) is a parallel interior-point solver for large-scale linear, quadratic and nonlinear programming that exploits the nested block structure often present in large-scale problems.

However, low-level languages such as Fortran or C++ have a rather steep learning curve and long write-compile-link-debug cycles. High-level scripting languages such as Python let the programmer do away with memory allocation concerns, and are typically far easier to learn and use than low-level compiled languages, which often outweighs the performance hit normally associated with them. Python is a full-fledged object-oriented programming language with a standard library that is nearly as extensive as that of C and C++. There has been much activity recently devoted to developing collections of tools for both researchers, modelers and practitioners in the Python programming language. We review those that, in our view, are prominent in the current landscape.

Pyomo (Hart et al., 2012) is a Python library for modeling optimization problems and is part of the Coopr (Hart, 2009) optimization repository. The syntax of Pyomo is strongly inspired by that of the AMPL modeling language and it currently supports linear, nonlinear and stochastic programs. In Pyomo, derivatives are computed via the ASL but it does not rely on availability of an AMPL engine. Rather, a Pyomo model is converted to a so-called *nl* file, representing computational graphs, which is then fed to the ASL (Gay, 2005).

PuLP (Stuart et al., 2011) is a modeling tool for linear programming that is able to export problems to MPS or LP format and subsequently call a linear programming solver such as GLPK (Makhorin, 2006), COIN, CPLEX or X-PRESS.

CVXOPT (Dahl and Vandenberghe, 2009) is a complete environment for modeling and solving convex problems, whether differentiable or not, by way of an interior-point method. CVXOPT uses its own interface to the BLAS for fast array and dense matrix operations and its own implementation of a sparse matrix library.

PyOpt (Perez et al., 2012) is a Python framework for formulating and solving optimization problems. Its goal is somewhat different from NLP.py as it provides an optimization framework with access to a variety of existing optimization algorithms accessible through a common interface. The focus is on formulating and solving nonlinear constrained optimization problems rather than developing new optimization algorithms.

Finally, Audet et al. (2010) propose the OPAL Python environment for modeling and solving non-smooth optimization problems with particular emphasis on algorithmic parameter optimization. OPAL interfaces a mesh-adaptive direct search method and provides modeling facilities to describe parameter-optimization problems and to assess performance by way of tailored combinations of atomic performance measures.

Python's notoriety is also apparent in major linear algebra libraries such as PETSc and TRILINOS, which offer Python bindings (Balay et al., 2009; Sala et al., 2008).

The rest of the paper is organized as follows: Section 7.2 describes the design philosophy of NLP.py, §7.3 summarizes the modeling facilities offered, §7.4 describes our in-house Cython sparse matrix library, §7.5 reviews the optimization and linear algebra building blocks available, §7.6 gives a brief overview of a few of the complete solvers written in NLP.py, and §7.7 shows examples illustrating the modularity of NLP.py. Finally, some concluding remarks are provided in §7.8.

7.2 Overall Design and Structure

NLP.py may be viewed as a collection of tools written in Python and interfaces to core libraries written with the Cython (Behnel et al., 2011) extension of Python. Cython is a superset of Python that facilitates interfacing with libraries that expose a C API and allows users to type variables. Because Cython code is compiled, it results in increased performance, yet remains easier to maintain and update than C or C++. Thanks to the strong connection between Python and Cython, core libraries appear transparently to the user as regular Python objects.

A solid sparse linear algebra library is essential if one is to solve large-scale sparse problems efficiently. The library of choice in NLP.py is CySparse (Arreckx et al., 2016b), but several other libraries are also supported to varying degrees. CySparse is described in §7.4.

The components of NLP.py revolve around the main tasks with which one is confronted in both modeling and algorithmic design. We now briefly review those tasks and describe them in more depth in the next sections.

Supplying derivatives The first important decision about a model is whether derivatives are available and in the affirmative, whether they will be implemented by hand, supplied by an automatic differentiation engine, or approximated using, say, finite differences or a quasi-Newton scheme.

Matrix storage schemes If it is anticipated that a solver will require Jacobians and/or Hessians, a user must decide how they should be stored—e.g., as dense arrays, sparse matrices in a specific storage format, or implicitly as linear operators. The choice of storage scheme is closely related to what types of operations a solver will need to perform with matrices, e.g., construct block saddle-point systems, and what linear algebra kernels will be used during the iterations.

Modeling Models are represented as Python objects that derive from one or two base classes. The object-oriented design of NLP.py dictates that a model results by inheritance from a model class defining the provenance of derivatives with another model class specifying a matrix storage scheme. For instance, multiple model classes obtain their derivatives from ADOL-C—one per matrix storage scheme—and all derive from a base class named `AdolcModel`. Similarly, multiple model classes store matrices according to one of the formats available in the CySparse library—one per derivative provider, and all derive from a base class named `CySparseModel`. For instance, a `CySparseAdolcModel` class could be defined as inheriting from both `AdolcModel` and `CySparseModel`.

Passing models to solvers The choice of a solver depends on multiple factors, including the matrix storage scheme, but more importantly, the structure of the problem. In NLP.py, several solvers are available and correspond to several types of problems. There are solvers for unconstrained optimization, linear and convex quadratic optimization, equality-constrained convex or nonconvex optimization, problems with complementarity constraints, and general problems featuring a mixture of equality constraints, inequality constraints and bounds. A main driver that may be called from the command line is supplied with each solver for problems written in the AMPL modeling language. The PyKrylov companion package supplies several iterative solvers for linear least-squares problems, including regularized problems and problems with a trust-region constraint.

Designing solvers Optimization researchers must often make implementation choices that strike a balance between performance, adherence to theoretical requirements and ease of implementation. An example that comes to mind is a linesearch-based method in which the

search must ensure satisfaction of the strong Wolfe conditions. Implementing such a linesearch is far more involved than implementing a simple Armijo backtracking search. The latter may not satisfy the assumptions necessary for convergence, but may still perform well in practice. Similarly, a trust-region method that relies on models using exact second derivatives may perform well, but what if a quasi-Newton approximation is used instead? What if we wish to terminate the iterations early if a condition depending on external factors is satisfied? What if we wish to experiment with a different merit function? It seems important for researchers to be able to experiment easily and swap a linesearch procedure for another, swap a model with exact second derivatives for a quasi-Newton model, and so forth.

The next sections elaborate on the above aspects.

7.3 Modeling

In NLP.py, a general optimization problem is formulated as

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to } \; c^{\text{L}} \leq c(x) \leq c^{\text{U}}, \quad \ell \leq x \leq u, \tag{7.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function, $c : \mathbb{R}^n \to \mathbb{R}^m$ is the (vector-valued) constraint function, and $c^{\text{L}} \in (\mathbb{R} \cup \{-\infty\})^m$, $c^{\text{U}} \in (\mathbb{R} \cup \{+\infty\})^m$, $\ell \in (\mathbb{R} \cup \{-\infty\})^n$, and $u \in (\mathbb{R} \cup \{+\infty\})^n$. Bounds on the variables appear separately from other types of constraints because numerical methods often treat them differently. The j-th general constraint is an equality constraint if $c^{\text{L}}_j = c^{\text{U}}_j$. Similarly, if $\ell_i = u_i$, the i-th variable is fixed and may technically be eliminated from the problem.

General optimization problems are represented using a subclass of the `NLPModel` abstract class. The main idea is that `NLPModel` acts as a placeholder for attributes and methods common to all optimization problems, and that a subclass specifies the structure of a problem of interest—e.g., an unconstrained problem, a bound-constrained problems, a quadratic problem, and so forth. An instance of the subclass gives life to the model by filling in the blanks. It does so by giving values to attributes such as the number of variables and of general constraints, arrays representing c^{L}, c^{U}, ℓ and u, an initial guess, and initial Lagrange multiplier estimates. The instance has methods for querying the model, including evaluating the objective function, the gradient of the objective, the (sparse) Jacobian matrix of the constraint functions, and the (sparse) Hessian matrix of the Lagrangian.

NLP.py predefines the `UnconstrainedNLPModel` and `BoundConstrainedNLPModel` subclasses of `NLPModel` that may be used as a shortcut to model unconstrained and bound-constrained problems. Listing 7.1 shows one way to implement the generalized Rosenbrock function for

an arbitrary value of n and how to create an instance with $n = 10$.

While Listing 7.1 seems intuitive we immediately hit severe limitation of this approach: we must hard code the derivatives of the objective and constraints functions if a solver is to make use of them. One might use finite-difference approximations but the order of convergence of descent methods often suffers from such approximations. A much more viable approach is to use automatic differentiation (AD) (Griewank, 2000). This is especially interesting in the large-scale case as the cost of evaluating derivatives via AD is typically a moderate multiple of the cost of evaluating the function itself. A number of generic automatic-differentiation packages are readily available—see `www.autodiff.org`. Some AD packages already have mature Python bindings and there also exist pure Python AD packages. NLP.py offers interfaces to ALGOPy (Walter, 2011a), PyADOLC (Walter, 2011b) and PyCppAD (Walter, 2011c).

Modeling with, e.g., PyADOLC is as simple as these authors could hope. Models in which derivatives should be computed by ADOL-C should be instances of a subclass of `AdolcModel`. Defining a class for unconstrained problems whose derivatives should be computed by ADOL-C (Walther et al., 2005) consists in inheriting from both `UnconstrainedNLPModel` and `AdolcModel`. The latter is itself a subclass of `NLPModel` that ensures that only functions need be supplied and subsequent derivatives will be evaluated behind the scenes by ADOL-C. Listing 7.2 illustrates the process without repeating operations already performed in Listing 7.1.

Listing 7.2 illustrates how multiple inheritance is used in NLP.py to mix base classes together so as to obtain models with the desired features. The same effect could be achieved by defining the class `AdolcRosenbrock` as inheriting from the `Rosenbrock` class of Listing 7.1 and from `AdolcModel`. The resulting `AdolcRosenbrock` model has dense second derivatives, but we obtain sparse Hessians by inheriting from `SparseAdolcModel` instead of `AdolcModel`. We obtain sparse Hessians in SciPy format by inheriting from `SciPyAdolcModel`, etc. The same principles may be used to define unconstrained quasi-Newton models, quadratic models in

```
from nlp.model.nlpmodel import UnconstrainedNLPModel
from numpy import sum
from numpy.random import random

class Rosenbrock(UnconstrainedNLPModel):
  def obj(self, x):
    return sum((1-x[:-1])**2 + 100*(x[1:]-x[:-1]**2)**2)

prob = Rosenbrock(10, name="Generalized Rosenbrock")
prob.obj(random(10))  # evaluate objective at a random point
```

Listing 7.1 Subclassing `UnconstrainedNLPModel` and creating an instance.

```
from nlp.model.adolcmodel import AdolcModel

class UnconstrainedAdolcModel(UnconstrainedNLPModel, AdolcModel):
  pass  # do nothing; the base classes do all the work

class AdolcRosenbrock(UnconstrainedAdolcModel):
  def obj(self, x):
    return np.sum((1-x[:-1])**2 + 100*(x[1:]-x[:-1]**2)**2)

prob = AdolcRosenbrock(10, name="Generalized Rosenbrock")
prob.obj(random(10))  # evaluate objective at a random point
prob.grad(random(10))  # evaluate gradient at a random point
prob.hess(random(10))  # evaluate Hessian at a random point
```

Listing 7.2 Inheriting from both `UnconstrainedNLPModel` and `AdolcModel`.

which the Hessian is stored in CySparse format, bound-constrained models in which derivatives are computed by CppAD, *etc.*

Another convenient approach implemented in NLP.py is to use the AMPL modeling language (Fourer et al., 2002) and leverage its mature automatic-differentiation features as most researchers and users of optimization are already familiar with it. Modeling in AMPL is simple and intuitive, and several standard benchmark collections are modeled in AMPL, e.g., (Dolan et al., 2004; Vanderbei, 2009). In NLP.py, AMPL models are used in the same way as `NLPModel`s and are generated from the *model* and *data* files constituting the AMPL model, or from the *nl* file decoded by the AMPL engine. Essentially, the *nl* file contains the expression tree of all linear and nonlinear expressions in the model and allows for automatic differentiation via the ASL (Gay, 1997).

In a similar vein, a subclass of `NLPModel` that represents a model decoded from a CUTEst problem (Gould et al., 2015b) is in development.

Lastly, PDE-constrained problems have been in the spotlight in recent years as challenging and highly structured. In our experience, PDE libraries have a steep learning curve and the optimization methods that they feature, if any, are few and often written to solve specific problems. The great divide between optimization libraries and PDE libraries makes it difficult for optimization research to benefit from testing on a large base of PDE-constrained problems and for PDE libraries to benefit from the latest advances in optimization. In the Python language, there is however a good match. FEniCS (Logg et al., 2012) is an extensive finite-element library with a full-featured Python interface named DOLFIN (Logg and Wells, 2010). DOLFIN allows the user to define domains, meshes, function spaces, finite-element families to approximate unknowns, and to model functionals and sets of PDEs in weak form with extraordinary ease. For given domain, mesh, function space and finite-element family, a functional is automatically discretized and it is possible to evaluate it as well as

its derivatives with respect to the unknown function. Similarly, a set of PDEs in weak form with accompanying boundary conditions is automatically discretized and its derivatives can also be obtained. That makes it possible to devise a generic modeling interface for PDE-constrained problems. In NLP.py, the latter has materialized as a subclass of `NLPModel` named `PDENLPModel`. In order to be instantiated, a `PDENLPModel` must be associated with a domain, a mesh, a function space, boundary conditions, as well as an optional initial guess, bounds on the unknown function, and possibly constraint left and right-hand sides. Before creating an instance, a user must subclass `PDENLPModel` in order to specify an objective functional and the constraints. PDE-constrained problems, and variational calculus problems in general, are a recent addition to NLP.py and will be the subject of a follow-up report. In the present paper, we show a simple example that illustrates the anatomy of such problems. Consider the distributed Poisson control problem with Dirichlet boundary conditions

$$\begin{aligned} \min_{u,f} \quad & \tfrac{1}{2}\int_\Omega |u(x) - u_0(x)|^2 \,\mathrm{d}x + \tfrac{1}{2}\beta \int_\Omega |f(x)|^2 \,\mathrm{d}x \\ \text{subject to} \quad & -\nabla \cdot (\nabla u(x)) = f(x) \quad x \in \Omega, \\ & u(x) = u_0(x) \quad x \in \partial\Omega, \end{aligned}$$

where $\beta > 0$ is a regularization parameter and u_0 is given. The implementation of the distributed control problem in NLP.py is given in Listing 7.3. The specification of the boundary conditions was left out for conciseness. In DOLFIN notation, `dot(f,f)*dx` represents the integral over the entire domain of $|f|^2$. Though not all details are shown, if `model` is an instance of the `DistributedControl` class, it is an `NLPModel` like any other, and one may call `model.obj()`, `model.grad()` and `model.hess()` as with any other model. Behind the scenes, DOLFIN is computing derivatives for us and the infrastructure set in `PDENLPModel` arranges so the model appears to the user as any other model.

The above examples show that NLP.py tries to attain the level of abstraction that is necessary in order to be able to write optimization solvers that are agnostic to the provenance of models,

```
1  class DistributedControl(PDENLPModel):
2    def register_objective_functional(self):
3      u, f = split(self.u)   # (u,f) lives in mixed-FEM space
4      du = u - self.target   # self.target is u0
5      return 0.5 * (dot(du,du)*dx + self.beta * dot(f,f)*dx)
6
7    def register_constraint_form(self):
8      u, f = split(self.u)
9      v, w = split(TestFunction(self.function_space))
10     return dot(grad(v),grad(u))*dx - v*f*dx
```

Listing 7.3 Implementation of a simple boundary control problem.

provided that they adhere to the (very general) interface described by `NLPModel`.

Any instance of any subclass of `NLPModel` may be passed directly to optimization algorithms in NLP.py or may be composed first with another class so as to apply problem transformations. An example situation where this is useful is when an algorithm expects to receive a problem with only equality constraints and bounds. The `SlackFramework` class does just this by inheriting from `NLPModel` and specializing the model by adding slack variables. This means that a `SlackFramework` object behaves exactly as an `NLPModel` object but the methods to evaluate the constraints, the bounds and the constraints Jacobian reflect the new form of the problem:

$$\min_{x \in \mathbb{R}^n, s \in \mathbb{R}^m} \; f(x) \quad \text{subject to } \; c(x) - s = 0, \; c^{\text{L}} \leq s \leq c^{\text{U}}, \; \ell \leq x \leq u. \tag{7.2}$$

A more elaborate example is the `AugmentedLagrangian` class, which derives from `NLPModel` and represents the bound-constrained proximal augmented Lagrangian problem whose (parametrized) objective function is

$$f(x) - y_k^T(c(x) - s) + \tfrac{1}{2}\rho_k(\|x - x_k\|^2 + \|s - s_k\|^2) + \tfrac{1}{2}\delta_k\|c(x) - s\|_2^2, \tag{7.3}$$

where $\rho_k \geq 0$ is a proximal parameter, $\delta_k > 0$ is a penalty parameter, $y_k \in \mathbb{R}^m$ represents a current approximation to the vector of Lagrange multipliers associated to the equality constraints of (7.2), and whose only constraints are the bounds in (7.2).

7.4 CySparse: A Fast Sparse Matrix Library

CySparse (Arreckx et al., 2016b) is a sparse matrix library written in Cython that can be used in Python and Cython projects. CySparse was written as a successor of PySparse (Geus et al., 2009) with the main goal of eliminating C libraries and the intricacies of interfacing them, and using instead libraries written in Python and Cython. CySparse's typed matrix indices and elements are fully compatible with NumPy (Oliphant, 2006) internal types.

The three sparse matrix data structures currently supported are modeled after PySparse: LL (Linked List), CSC (Compressed Sparse Column) and CSR (Compressed Sparse Row). SciPy (Jones et al., 2001) offers corresponding structures. We compare them in section 7.4.1.

A specialized implementation of each data structure exists for each index and element type. In addition, each data structure provides the option to store explicit zeros. Only the lower triangle of symmetric matrices is stored. With two index types and seven element types, there are 84 combinations. In order to specialize code for each combination, we created our own

code generation tool based on the Jinja2 (Ronacher, 2008) templating engine and packaged it as the cygenja (Arreckx et al., 2016a) standalone library, which may be reused in any Cython project. NumPy and SciPy use similar tools to generate their own typed versions of templated code. When Cython's *fused types* mechanism has matured, the templating engine can technically be removed from CySparse, which should reduce its footprint.

One of the strengths of CySparse is the frequent use of *lazy evaluation*. Certain operations such as matrix addition and multiplication, and multiplication by a scalar are delayed until an end result is required. The matrix product AB of two sparse matrices is not computed until the user explicitly requests a result, such as the product itself, the application of AB to a vector, the (i, j)-th element of AB, etc.

LL matrices have a *view* attached to them—a proxy data structure that corresponds to selected parts of it. Views are pointers and consume little memory. Therefore, extracting submatrices from a given matrix is a cheap operation. Similarly, associated matrices, such as the transposed, conjugate and transpose conjugate, are proxies and can be used almost anywhere a real matrix can.

7.4.1 Benchmarks

We provide a few benchmarks to showcase the performance of CySparse. We compare it with PySparse (Geus et al., 2009) and SciPy and we only discuss multiplication of a sparse matrix with a dense NumPy vector. We randomly generate square sparse matrices of size $n \times n$ with nnz nonzero elements. We generate dense NumPy vectors with NumPy's `arange(0, n, dtype=np.float64)`.

For each benchmark, we run the four scenarii described in Table 7.1 100 times for each operation. For each scenario, operations are ranked by runtime. The most efficient implementation gets a value of 1.0. The values of the other operations are relative, i.e. an operation with value k takes k times as long to execute as the most efficient operation.

CySparse, PySparse and SciPy, were compiled and tested with the same flags on the same

Table 7.1 Four benchmark scenarii.

Scenario	n	nnz
1	10^4	10^3
2	10^5	10^4
3	10^6	10^5
4	10^6	$5 \cdot 10^3$

machine and using `int32` indices and `float64` elements.

Table 7.2 reports benchmarks on the basic `matvec` operation, i.e. the multiplication Av where A is a $n \times n$ sparse matrix and v is a NumPy vector of length n.

CySparse lets the user simply type `A*v` to compute the product. PySparse does not support the notation `A*v` and only offers `A.matvec(v,y)` where `y` is a preallocated vector to store the result. Because both SciPy and CySparse allocate such a vector transparently, we take into account the time required by PySparse to allocate `y`.[1] CySparse and CySparse 2 in Table 7.2 represent `y = A*v` and `y = A.matvec(v)` respectively while SciPy and SciPy 2 represent `y = A*v` and `y = A._mul_vector(v)`, respectively. The "2" variants are equivalent to `A*v` minus a small convenience layer that permits the shorthand notation.

Table 7.2 reveals that using LL and CSR formats, CySparse and PySparse are on par while SciPy is slightly faster than CySparse when using the CSC format.

The second benchmark in Table 7.3 investigates the case where the dense NumPy vector is not contiguous in memory. We can see that our specialized implementation pays off.

The third and last benchmark in Table 7.4 compares the multiplication of two sparse matrices with a dense NumPy vector. CySparse computes $A \cdot B \cdot v$ as $A \cdot (B \cdot v)$ and clearly this is faster than first computing $A \cdot B$ and then $(A \cdot B) \cdot v$. Even when we force SciPy to compute $A \cdot (B \cdot v)$, CySparse remains slightly faster.

7.5 Building Blocks for Optimization

7.5.1 Globalization Strategies

One of the most important ingredients in nonlinear optimization is the globalization strategy, i.e., the mechanism by which locally-convergent methods, such as Newton's method, can converge from a remote initial guess. Such mechanisms essentially come in two flavors: the linesearch and the trust region. We examine them in turn and how they are implemented in NLP.py.

In constrained optimization, progress is often measured by way of a *merit function* which is typically a weighted combination of the objective and constraint functions. Examples include ℓ_p-norm exact merit functions, the augmented Lagrangian function, the logarithmic barrier function or the objective function itself in unconstrained optimization. We refer the interested reader to, e.g., (Fletcher, 1987) for further information.

1. Using `y = numpy.empty(n, dtype=numpy.float64)`.

Table 7.2 Sparse matrix with dense contiguous vector multiplication

`y = A*v` with `A` a LL sparse matrix							
Scenario 1		Scenario 2		Scenario 3		Scenario 4	
PySparse	1.000	PySparse	1.000	CySparse 2	1.000	PySparse	1.000
CySparse 2	1.158	CySparse 2	1.023	CySparse	1.023	CySparse	1.043
CySparse	1.220	CySparse	1.032	PySparse	1.045	CySparse 2	1.064
SciPy 2	86.395	SciPy	101.536	SciPy 2	55.690	SciPy	133.495
SciPy	87.267	SciPy 2	102.131	SciPy	56.398	SciPy 2	135.512
`y = A*v` with `A` a CSR sparse matrix							
Scenario 1		Scenario 2		Scenario 3		Scenario 4	
PySparse	1.000	CySparse 2	1.000	CySparse	1.000	PySparse	1.000
CySparse 2	1.178	CySparse	1.022	PySparse	1.000	CySparse	1.018
CySparse	1.212	PySparse	1.037	CySparse 2	1.009	CySparse 2	1.061
SciPy 2	1.333	SciPy 2	1.287	SciPy 2	1.114	SciPy	1.419
SciPy	1.373	SciPy	1.308	SciPy	1.135	SciPy 2	1.421
`y = A*v` with `A` a CSC sparse matrix							
Scenario 1		Scenario 2		Scenario 3		Scenario 4	
SciPy 2	1.000	SciPy	1.000	SciPy 2	1.000	SciPy	1.000
SciPy	1.102	SciPy 2	1.004	SciPy	1.002	SciPy 2	1.000
CySparse 2	1.107	CySparse	1.084	CySparse	1.059	CySparse	1.138
CySparse	1.146	CySparse 2	1.120	CySparse 2	1.060	CySparse 2	1.148

Table 7.3 CSC sparse matrix with dense non contiguous vector multiplication

Scenario 1		Scenario 2		Scenario 3		Scenario 4	
CySparse 2	1.000	CySparse 2	1.000	CySparse 2	1.000	CySparse	1.000
CySparse	1.011	CySparse	1.098	CySparse	1.006	CySparse 2	1.007
SciPy 2	1.354	SciPy	2.273	SciPy 2	1.733	SciPy 2	3.193
SciPy	1.394	SciPy 2	2.286	SciPy	1.742	SciPy	3.216

Table 7.4 Sparse matrix with dense contiguous vector multiplication where the sparse matrix is obtained as the product of two sparse matrices

`y = A*B*v` with `A` a CSR sparse matrix, B a CSC sparse matrix and `v` a dense NumPy vector							
Scenario 1		Scenario 2		Scenario 3		Scenario 4	
CySparse	1.000	CySparse	1.000	CySparse	1.000	CySparse	1.000
SciPy	5.386	SciPy	3.921	SciPy	3.850	SciPy	5.207
`y = A*(B*v)` with `A` a CSR sparse matrix, B a CSC sparse matrix and `v` a dense NumPy vector							
Scenario 1		Scenario 2		Scenario 3		Scenario 4	
CySparse	1.000	CySparse	1.000	CySparse	1.000	CySparse	1.000
SciPy	1.019	SciPy	1.016	SciPy	1.011	SciPy	1.049

Let ψ denote a merit function. In linesearch methods, a search direction d is identified from the current iterate x and a steplength $t > 0$ is sought so as to produce a *sufficient* decrease in the value of ψ. The notion of sufficient decrease varies with the context and the smoothness of the merit function. When ψ is continuously differentiable, typical conditions on t are the *Armijo condition*

$$\psi(x + td) \leq \psi(x) + \alpha t \nabla\psi(x)^T d, \tag{7.4}$$

where $0 < \alpha < 1$ is a parameter, or the *strong Wolfe conditions*, which additionally require that

$$|\nabla\psi(x + td)^T d| \leq \sigma |\nabla\psi(x)^T d|, \tag{7.5}$$

where $\alpha < \sigma < 1$.

In NLP.py, a linesearch is represented by an abstract `LineSearch` class and is initialized by first restricting a model to a line, which is done by creating an instance of the `C1LineModel` class. If `model` represents a problem with objective f and constraints c, if `x` is the current iterate and `d` is a nonzero vector, then `C1LineModel(model, x, d)` represents a model with objective $\phi(t) := f(x + td)$ and constraints $\gamma(t) := c(x + td)$ as well as bounds on t computed from bounds in `model`, `x` and `d`. Strictly speaking, the restricted model may be infeasible unless x is feasible for the original model.

Two types of linesearch are currently available in NLP.py: The Armijo linesearch and the strong Wolfe linesearch as implemented by Moré and Thuente (1994). In addition, a modified Armijo linesearch that initially increases the step in hopes to satisfy the Wolfe conditions is used in our Python implementation of the limited-memory BFGS method described in §7.6.

The second type of globalization mechanism available in NLP.py is the trust region. In a

trust-region method for unconstrained optimization, a model $m(x+d)$ of the objective f about the current iterate x is approximately minimized within a compact *trust region* containing x. Typically, this region is a Euclidean ball or a box centered at x and the model is a quadratic agreeing with f at x at least up to first order. A typical trust-region problem is thus to

$$\min_{d \in \mathbb{R}^n} \; m(x+d) \quad \text{subject to } \|d\| \leq \Delta. \tag{7.6}$$

In constrained optimization, the model is often a simplified version of a merit function. We refer the reader to the book of Conn et al. (2000) for more information. In the simplest case, building a quadratic model of f is done by creating an instance `QPModel(g, H)` where `g` is a vector specifying the linear term and `H` is a symmetric matrix or linear operator specifying the quadratic term, but note that it is possible for the quadratic model to have constraints. The abstract `TrustRegionFramework` class encapsulates the details pertaining to the definition of a trust region and to its management, including the computation of the ratio of achieved to predicted reduction. For flexibility, the definition of the model is left to the user. A `TrustRegionFramework` is initialized with an initial trust-region radius and constants related to its management and to step acceptance.

7.5.2 Numerical Linear Algebra

In this section, we briefly describe several companion packages that combine with NLP.py and provide linear algebra tools of critical importance to optimization.

Ordering and scaling The HSL.py satellite package (Arreckx et al., 2016c) provides flexible access to ordering and scaling methods from the Harwell Subroutine Library (2007), including the profile and wavefront reducing orderings of MC60, the row-permutation package MC21, which aims to produce a matrix with as many nonzeros as possible on the main diagonal, and the scaling package MC29.

Factorization of Symmetric Matrices NLP.py provides access to several libraries for the solution of sparse symmetric linear systems. The Harwell Subroutine Library (2007) subroutines MA27 of Duff and Reid (1982) and MA57 of Duff (2004) implement multifrontal sparse symmetric indefinite factorizations and are flexible enough to allow the factorization of definite, symmetric indefinite, and symmetric quasi-definite (Vanderbei, 1995) matrices. HSL.py interfaces both MA27 and MA57 via the common generic class `Sils`.[2] Two specialized classes `MA27Solver` and `MA57Solver` provide seamless access to the factorization, solution

2. Symmetric Indefinite Linear System

and iterative refinement routines, and to statistics on the factorization, including the inertia of the matrix, and in particular its number of negative eigenvalues—a critical information when factorizing saddle-point matrices (Gould, 1985).

MUMPS (Amestoy et al., 1998) is a multifrontal factorization for both symmetric and unsymmetric matrices that targets distributed memory computers, features out-of-core factorization, and accomodates sparse right-hand sides. MUMPS.py provides a Python interface to MUMPS that is compatible with CySparse matrices. Because MUMPS.py and HSL.py share the same solver interfaces, MA27, MA57 and MUMPS may be simply swapped when used in NLP.py.

Factorization of Non-Symmetric Matrices NLP.py can be used in conjunction with the satellite packages qr_mumps.py and SuiteSparse.py to factorize rectangular matrices, which is crucial to least-squares problems, as well as square unsymmetric matrices.

qr_mumps.py is a set of interfaces to qr_mumps (Buttari, 2013), a multicore QR factorization well suited to the solution of large and sparse least-squares and least-norm problems. In the same vein, SuiteSparse.py provides access to the multifrontal sparse LU factorization implemented in UMFPACK (Davis, 2004) and the supernodal sparse Cholesky factorization implemented in CHOLMOD.

The Python classes exposing those packages accept a sparse matrix in coordinate format or a CySparse matrix. Therefore, any CySparse matrix originating from NLP.py may be passed to those classes and factorization methods can be easily interchanged.

Iterative Methods

PyKrylov (Orban, 2009) is a pure Python implementation of a number of Krylov subspace methods for symmetric and non-symmetric linear systems together with a library of *linear operators*, i.e., abstract objects that derive from a base `LinearOperator` class and encapsulate the action of a linear operator on a vector. They obey the laws of mathematical linear operators, i.e., $(A + B)x = Ax + Bx$ and $(\alpha A)x = \alpha(Ax)$. Linear operators may be used to wrap matrices and linear functions, and may be chained by way of multiplication. For example, if `A` and `B` are two linear operators, `C=A*B` is another linear operator but its construction is cheap; only its action `C*x` is computed when requested, and it will be computed as `A*(B*x)`. Linear operators may be transposed, conjugated, added together, restricted or composed into block operators, making them ideal tools to work with in the context of iterative methods for linear systems.

PyKrylov provides access to limited-memory quasi-Newton operators in standard or compact

form (Byrd et al., 1994), including the limited-memory BFGS, DFP and SR1 approximations.

Because the most expensive operations in Krylov methods are operator-vector products, dot products and *axpy*s, the performance hit incurred by PyKrylov for being implemented in a high-level language is low. Assuming the user implements efficient operator-vector products, by relying on a fast sparse matrix package or otherwise, NumPy handles vector operations efficiently via an optimized BLAS. A considerable advantage of pure Python implementations of Krylov methods is that adding new methods is simple via subclassing. PyKrylov currently features the conjugate gradient algorithm (Hestenes and Stiefel, 1952), the bi-conjugate gradient stabilized algorithm (van der Vorst, 1992), the symmetric LQ method (Paige and Saunders, 1975), the transpose-free quasi-minimum residual algorithm (Freund, 1993), the conjugate gradient squared algorithm (Sonneveld, 1989), MINRES (Paige and Saunders, 1975), LSMR (Fong and Saunders, 2011), LSQR (Paige and Saunders, 1982b) and CRAIG (Saunders, 1995). PyKrylov is distributed separately from the main NLP.py source code as it is useful in other contexts.

Other iterative methods are part of NLP.py itself because of their relevance to optimization. Those include the truncated conjugate-gradient algorithm of Steihaug (1983), and the projected conjugate gradient algorithm of Gould et al. (2001).

Preconditioning A good preconditioner is often essential in the iterative solution of linear systems or trust-region subproblems. The diversity of applications of optimization makes it difficult to supply useful specialized preconditioners. For this reason, a few generic preconditioners are available in NLP.py. There are currently three types of preconditioners available: diagonal, band and based on the limited-memory LDL^T factorization.

A diagonal preconditioner is simple and only consists in extracting the diagonal from an explicit matrix. If A is a square matrix, the diagonal preconditioner is D^{-1} where $d_{ii} = \max\{|a_{ii}|, 1\}$. Band preconditioners require one of the factorizations for symmetric definite matrices covered in §7.5.2. More efficient factorizations exist for banded systems, such as that of Gill et al. (1981) and of Schnabel and Eskow (1999). Incomplete factorizations are popular preconditioners in a variety of fields such as multigrid methods for partial-differential equations. They typically consist in computing a factorization of a matrix, dropping elements in the factors that fall below a specified threshold in absolute value or that exceed the amount of memory the user is prepared to expend. This has the effect of promoting sparse factors, at the expense of preconditioner quality. A generalization of the incomplete Cholesky factorization of Lin and Moré (1999) to symmetric quasi-definite matrices is proposed by Orban (2014) and implemented in the LLDL.py package (Arreckx et al., 2016d).

7.6 Solvers

NLP.py provides complete solvers for various areas of optimization. Each solver can be run programmatically or as a stand-alone executable from the command line to solve problems in AMPL format. The intent is that solver classes be subsequently called from other applications in which optimization problems must be solved. In this section, we briefly describe the solvers currently available and the problem classes to which they apply, and highlight implementation specifics.

Unconstrained Optimization Unconstrained optimization problems have the general form

$$\min_{x \in \mathbb{R}^n} \; f(x), \tag{7.7}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is generally assumed to be once or twice continuously differentiable. An important distinction between numerical methods for (7.7) is based on the availability of second-order derivatives. The numerical methods for unconstrained optimization currently implemented in NLP.py reflect this distinction.

TRUNK, is a factorization-free non-monotone trust-region method. At each iteration, the second-order model

$$m_k(x_k + s) = f(x_k) + \nabla f(x_k)^T s + \tfrac{1}{2} s^T \nabla^2 f(x_k) s \approx f(x_k + s)$$

is approximately minimized inside a Euclidean trust region by way of the truncated conjugate gradient algorithm of Steihaug (1983). Instead of shrinking the trust-region radius on unsuccessful iterations, a backtracking linesearch is performed along the step in the spirit of Nocedal and Yuan (1998). The truncated conjugate gradient may be preconditioned with a user-supplied preconditioner such as a band preconditioner or a limited-memory Cholesky factorization—see §7.5.2. The convergence of TRUNK is covered in (Conn et al., 2000).

The second method implements the limited-memory BFGS algorithm of Liu and Nocedal (1989). An approximation of the Hessian matrix is implicitly maintained as a limited-memory BFGS matrix H_k such that $H_k^{-1} \approx \nabla^2 f(x_k)$. Global convergence is promoted by way of a strong Wolfe linesearch, but in practice we have found that a modified Armijo linesearch is nearly as effective. The management of H_k is confined to a linear operator defined in PyKrylov so that products of the form $H_k^{-1} d$ take the abstract form `H*d`. Internally, the product is computed by the two-loop recursion formula (Nocedal, 1980) or using the compact storage of limited-memory matrices (Byrd et al., 1994). The main class defining the framework for the limited-memory BFGS method has as one of its members an `InverseLBFGSOperator` object

whose role is to manage the circular stacks and compute matrix-vector products with H_k^{-1}. This modular design allows `InverseLBFGS` objects to be used in other contexts, such as for preconditioning.

Convex Quadratic Programming Interior-point methods are a well-established class of methods that have proved to be especially efficient on linear and convex quadratic programs, i.e., problems of the form

$$\min_{x \in \mathbb{R}^n} \; g^T x + \tfrac{1}{2} x^T H x \quad \text{subject to } Ax = b, \; x \geq 0, \tag{7.8}$$

where $g \in \mathbb{R}^n$, $H = H^T \in \mathbb{R}^{n \times n}$ is positive semi-definite, $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. When $H = 0$, (7.8) is a linear program. Note that a linear or convex quadratic program modeled for solution in NLP.py need not be in standard form, although (7.8) is written in this way for simplicity. Two of the best-known interior-point methods for (7.8) are the long-step method (Kojima et al., 1993) and the predictor-corrector method of Mehrotra (1992) in augmented form (Fourer and Mehrotra, 1993). To account for situations where A is (numerically) rank deficient, we implement the primal-dual regularized variant of Friedlander and Orban (2012). At each iteration, a quasi-definite linear system with coefficient of the form

$$\begin{bmatrix} -(X^{-1}Z + \rho I) & A^T \\ A & \delta I \end{bmatrix} \tag{7.9}$$

is solved, for iteration-dependent values ρ and δ decreasing to zero. Factorizing the latter matrix is typically substantially cheaper and faster than factorizing the indefinite augmented matrix with $\rho = \delta = 0$. Our implementation features row and column equilibration of A prior to solution which in our experience increases robustness and the accuracy of the linear system solution.

In NLP.py, the main abstract class implementing the primal-dual-regularized interior-point solver is named `InteriorPointSolver`. A user may elect to use the scaling implemented in the Harwell Subroutine Library (2007) subroutine MC29 instead of the row and column equilibration. In our implementation, this is realized by subclassing `InteriorPointSolver` and overriding the `scale()` and `unscale()` methods.

Dehghani and Orban (2016) subclass `InteriorPointSolver` to implement a corresponding method for linear least-squares problems with linear constraints without forming the normal equations operator.

Bound-Constrained Optimization Bound-constrained problems have the form

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to } \ell \leq x \leq u, \tag{7.10}$$

and occur naturally in numerous practical applications such as image reconstruction and the discretization of optimal control problems with obstacles, but they also occur as subproblems in methods for general nonlinear problems, such as augmented-Lagrangian methods. In (7.10), the function f is not assumed to be convex but its second derivatives are assumed to exist and be continuous.

We elected to implement TRON (Lin and Moré, 1998), an active-set method that iteratively determines a current working set by way of a projected gradient method, and explores faces of the feasible set using a Newton trust-region method. Our implementation is factorization-free in the sense that only Hessian-vector products are required, which allows us to use quasi-Newton approximations when second-order derivatives are not available or costly to obtain. When used with limited-memory BFGS approximations, TRON becomes a trust-region variant of L-BFGS-B (Byrd et al., 1995).

Equality-Constrained Optimization NLP.py features an implementation of the funnel method of Gould and Toint (2008) for the general equality-constrained problem

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to } c(x) = 0. \tag{7.11}$$

Funnel avoids the use of penalty parameters or filters and make use of two trust-region mechanisms to promote global convergence. Steps are computed as a combination of normal and tangential components. An example of opportunity for specialization occurs in the normal step computation, which requires the solution of an inconsistent underdetermined linear least-squares problem subject to a trust-region constraint. Such a problem can be solved using one of the least-squares solvers available in PyKrylov, and various least-squares solvers can be used via subclassing.

General Constrained Optimization In general constrained optimization, we seek to solve (7.1) in its most general form, i.e., the objective function is nonconvex as are the constraint functions. For simplicity of notation and because the numerical method described in this section treats bound constraints and general inequality constraints alike, we rewrite the problem as

$$\min_{x \in \mathbb{R}^n} \; f(x) \quad \text{subject to } c_{\mathcal{E}}(x) = 0, \quad c_{\mathcal{I}}(x) \geq 0, \tag{7.12}$$

where $\mathcal{E}$ and $\mathcal{I}$ are finite index sets with $n_{\mathcal{E}}$ and $n_{\mathcal{I}}$ elements, respectively. Note that (7.1) can always be cast as (7.12).

The first method we elected to implement as part of NLP.py for the solution of general constrained optimization problems consists in a mixed interior/exterior penalty method. The formulation (7.12) is transformed by using an ℓ_1-penalty function to penalize infeasibility. The nonsmooth penalty terms are subsequently rewritten as smooth terms by adding *elastic variables* s to the problem. The penalized problem takes the form

$$\min_{x \in \mathbb{R}^n,\ s \in \mathbb{R}^{n_{\mathcal{C}}}} \ \phi(x, s; \nu) \quad \text{subject to } \ c_i(x) + s_i \geq 0, \quad s_i \geq 0, \quad \text{for all } i \in \mathcal{C}, \tag{7.13}$$

where

$$\phi(x, s; \nu) = f(x) + \nu \sum_{i \in \mathcal{E}} (c_i(x) + 2s_i) + \nu \sum_{i \in \mathcal{I}} s_i.$$

In the above formulation, $\mathcal{C} = \mathcal{E} \cup \mathcal{I}$, $n_{\mathcal{C}} = n_{\mathcal{E}} + n_{\mathcal{I}}$, and $\nu > 0$ is a penalty parameter that is updated at each iteration. The strong relationship between (7.12) and (7.13) makes this approach attractive. Moreover, the elastic variables have a regularizing effect in the sense that (7.13) always satisfies a constraint qualification even if (7.12) does not. For this reason, Coulibaly and Orban (2012) apply the same idea to problems with complementarity constraints. We refer the interested reader to (Gould et al., 2015a) for more details. The `ElasticFramework` abstract class performs the transformation above behind the scenes using a derived class of `NLPModel`.

The second method for (7.12) that is implemented is an augmented Lagrangian method. Firstly slack variables are introduced so the problem has constraints of the form $c(x) = 0$ and $\ell \leq x \leq u$. The k-th outer iteration of the augmented-Lagrangian algorithm consists in approximately solving the bound constrained subproblem

$$\min_{x \in \mathbb{R}^n} \ f(x) + c(x)^T y_k + \tfrac{1}{2}\delta_k \|c(x)\|_2^2 \quad \text{subject to } \ \ell \leq x \leq u, \tag{7.14}$$

where y_k is the current vector of Lagrange multipliers estimates and $\delta_k > 0$. Subproblem (7.14) is solved using TRON with a limited-memory structured quasi-Newton Hessian approximation, i.e., one of the form $B_k + \delta_k J(x_k)^T J(x_k)$ where $B_k \approx \nabla_{xx} L(x_k, y_k)$, and where $L(x, y) = f(x) - c(x)^T y$ is the Lagrangian of the problem and $J(x)$ is the Jacobian of c at x. This implementation of the augmented Lagrangian resembles that in LANCELOT (Gould et al., 2003a) with the exception of the solution of (7.14). Arreckx et al. (2015) use this augmented Lagrangian in the context of high-fidelity structural-design optimization.

7.7 Applications

In this final section, we illustrate a few concepts from the previous sections via two examples.

Quasi-Newton TRON The first example is a customization of the TRON solver described briefly in §7.6. TRON, in the original version of Lin and Moré (1998), minimizes a sequence of quadratic models using the conjugate gradient method with an incomplete Cholesky factorization preconditioner. For this reason, it relies on exact second derivatives stored as an explicit matrix. In NLP.py, the default `TRON` class only assumes that Hessian-vector products are available and thus, applies the conjugate gradient method without preconditioner. Listing 7.4 shows how to define a subclass `QNTRON` of the `TRON` base class to handle models that employ a quasi-Newton Hessian approximation. In NLP.py, all solver classes possess a callback method named `post_iteration()` that is called, as the name indicates, at the end of every iteration. It allows users to perform additional tasks such as updating a limited-memory quasi-Newton approximation. After a new iterate is computed, the oldest vector pair in the set of pairs $\{s_i, y_i\}$, for $i = 1, ..., m$ is replaced by the new pair $\{\texttt{dvars}, \texttt{dgrad}\}$ computed from the most recent step. The rest of Listing 7.4 illustrates the creation of a model using the compact representation of a symmetric rank one approximation of the Hessian and the instantiation of a `QNTRON` solver that uses a truncated conjugate gradient to solve trust-region subproblems. The solver is started by calling its `solve()` method.

```
from pykrylov.linop import CompactLSR1Operator as LSR1
from nlp.optimize.tron import TRON
from nlp.optimize.pcg import TruncatedCG
from nlp.model.amplmodel import QNAmplModel as Model

class QNTRON(TRON):
    """A variant of TRON with L-SR1 quasi-Newton Hessian."""

    def __init__(self, *args, **kwargs):
        super(QNTRON, self).__init__(*args, **kwargs)
        self.save_g = True

    def post_iteration(self, **kwargs):
        # Update quasi-Newton approximation.
        if self.step_accepted:
            self.model.H.store(self.dvars, self.dgrad)

model = Model(problem, H=LSR1, npairs=5, scaling=True)

tron = QNTRON(model, TruncatedCG)
tron.solve()
```

Listing 7.4 Subclassing `TRON` for quasi-Newton approximations.

The performance profiles of Dolan and Moré (2002) are employed to compare the impact of using quasi-Newton approximations with TRON from the points of view of efficiency and robustness. Efficiency and robustness rates are readable respectively on the left and right vertical axes of the profile. Figure 7.1 compares the basic version of TRON, which uses exact second derivatives, with two limited-memory symmetric rank one versions, using 5 and 10 pairs in history on 124 unconstrained problems and 61 bound-constrained problems from the AMPL/CUTEr collection (Vanderbei, 2009). The plots indicate the performance loss that may be expected when using limited-memory SR1 approximations to the second derivatives. They show however that the compact representation and two-loop recursion implementations perform very similarly, and that there does not seem to be a definite advantage to using 10 pairs instead of 5.

Lagrange multipliers estimates in Funnel In the funnel method of Gould and Toint (2008), a sequence of normal and tangential steps are performed either to reduce constraint violation or the objective function while retaining the improvement in constraint violation. In preparation for a tangential step, new local Lagrange multiplier estimates y_k are computed by solving the least-squares problem

$$\min_y \tfrac{1}{2}\|g_k^N + J_k^T y\|^2,$$

where g_k^N is the gradient of a quadratic model of the objective function and J_k is the Jacobian of the constraints at x_k. In practice, one can compute such an approximation by applying a Krylov method such as LSQR (Paige and Saunders, 1982a,b), which is available in PyKrylov, starting from $y = 0$. Listing 7.5 illustrates how to implement this strategy by defining a subclass `LSQRFunnel` of the `Funnel` base class. Our subclass must implement the `multipliers_estimate()` method to provide Lagrange multipliers estimates.

```
from pykrylov.lls.lsqr import LSQRFramework
from nlp.optimize.funnel import Funnel

class LSQRFunnel(Funnel):

    def multipliers_estimate(self, A, b):
        LSQR = LSQRFramework(A)
        LSQR.solve(b, show=False)
        return (LSQR.x, LSQR.xnorm)
```

Listing 7.5 Subclassing `Funnel` for estimation of the Lagrange multipliers using LSQR.

If the user would now like to compute multipliers using a multi-core sparse QR factorization, the procedure is the same, and consists in subclassing `Funnel` and overloading the

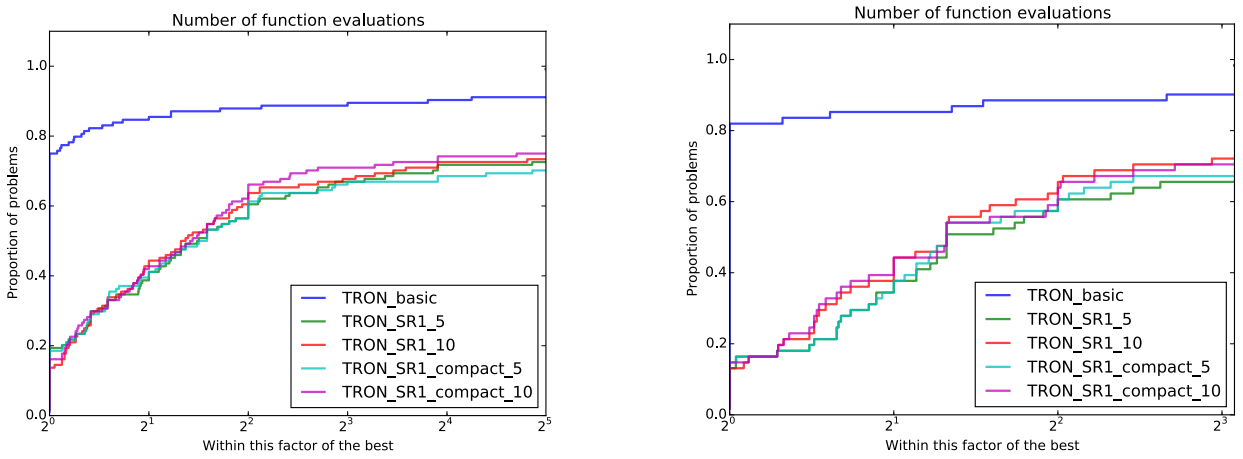

Figure 7.1 Performance profiles comparing the TRON with exact second derivatives and with symmetric rank one approximations using 5 or 10 pairs. Left: 124 unconstrained problems. Right: 61 bound-constrained problems.

`multipliers_estimate()` method. That is illustrated in Listing 7.6.

```
1  from qr_mumps.solver import QRMUMPSSolver
2  from nlp.optimize.funnel import Funnel
3
4  class QRMUMPSFunnel(Funnel):
5
6      def multipliers_estimate(self, A, b):
7          qr = QRMUMPSSolver(A)
8          qr.factorize()
9          x = qr.solve(b)
10         return (x, numpy.linalg.norm(x))
```

Listing 7.6 Subclassing `Funnel` for estimation of the Lagrange multipliers using qr_mumps.

7.8 Conclusion

The design of the NLP.py programming environment for optimization makes extensive use of object-oriented features, such as abstract classes and multiple inheritance. The environment provides basic building blocks for large-scale computational optimization. The Python language is mature but in constant evolution and scientific extensions to the language have grown for the past 15 years from basic interfaces to extensive state-of-the-art libraries.

Admittedly, we had to make choices as to which complete solvers to include in NLP.py. Numerous other solvers are promising candidates and will certainly be included in future versions, as will other sets of tools. At the present time, our hope is that researchers find NLP.py to be a valuable development platform for novel optimization methods and that the list

of tools and solvers expands. NLP.py may be obtained from `github.com/PythonOptimizers/NLP.py.git`.

REFERENCES

P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent (1998). Multifrontal parallel distributed symmetric and unsymmetric solvers. *Computer Methods in Applied Mechanics and Engineering*, 184:501–520.

S. Arreckx, A. Lambe, J. R. R. A. Martins, and D. Orban (2015). A matrix-free augmented Lagrangian algorithm with application to large-scale structural design optimization. *Optimization and Engineering*, pages 1–26. doi: 10.1007/s11081-015-9287-9.

S. Arreckx, D. Orban, and N. van Omme (2016a). Cygenja: A source code generator using Jinja2 templates. `github.com/PythonOptimizers/cygenja`.

S. Arreckx, D. Orban, and N. van Omme (2016b). CySparse: A Python/Cython library for sparse matrices. `github.com/PythonOptimizers/cysparse`.

S. Arreckx, D. Orban, and N. van Omme (2016c). HSL.py: A Python/Cython interface to the Harwell Subroutine Library. `github.com/PythonOptimizers/HSL.py`.

S. Arreckx, D. Orban, and N. van Omme (2016d). LLDL.py: A limited-memory ldl^t factorization in Python. `github.com/PythonOptimizers/LLDL.py`.

C. Audet, C.-K. Dang, and D. Orban (2010). Algorithmic parameter optimization of the DFO method with the OPAL framework. In J. C. K. Naono, K. Teranishi and R. Suda, editors, *Software Automatic Tuning: From Concepts to State-of-the-Art Results*, pages 255–274. Springer, New-York, NY.

S. Balay, K. Buschelman, W. D. Gropp, D. Kaushik, M. G. Knepley, L. C. McInnes, B. F. Smith, and H. Zhang (2009). PETSc Web page. `www.mcs.anl.gov/petsc`.

S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. S. Seljebotn, and K. Smith (2011). Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31–39. doi: 10.1109/MCSE.2010.118.

S. Benson, L. C. McInnes, J. J. Moré, T. Munson, and J. Sarich (2016). TAO user manual (revision 3.7). Technical Report ANL/MCS-TM-322, Argonne National Laboratory, Argonne, Illinois, USA. `www.mcs.anl.gov/tao`.

A. Buttari (2013). Fine-grained multithreading for the multifrontal QR factorization of sparse matrices. *SIAM Journal on Scientific Computing*, 35(4):C323–C345. doi: 10.1137/110846427.

R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu (1995). A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208. doi: 10.1137/0916069.

R. H. Byrd, J. Nocedal, and R. B. Schnabel (1994). Representations of quasi-Newton matrices and their use in limited memory methods. *Mathematical Programming*, 63(1):129–156. doi: 10.1007/BF01582063.

A. R. Conn, N. I. M. Gould, and Ph. L. Toint (2000). *Trust-Region Methods*. SIAM, Philadelphia, USA.

Z. Coulibaly and D. Orban (2012). An ℓ_1 elastic interior-point method for mathematical programs with complementarity constraints. *SIAM Journal on Optimization*, 22(1):187–211. doi: 10.1137/090777232.

P.-R. Curatolo (2008). Méthodes de pénalisation pour l'optimisation de structures. Ms thesis, École Polytechnique de Montréal, Montréal, Québec, Canada.

J. Dahl and L. Vandenberghe (2009). CVXOPT: Python software for convex optimization. `abel.ee.ucla.edu/cvxopt`.

T. A. Davis (2004). Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199. doi: 10.1145/992200.992206.

M. Dehghani and D. Orban (2016). A regularized interior-point method for constrained linear least squares. Cahier du GERAD G-2016-xx, GERAD, Montréal, QC, Canada. In preparation.

E. Dolan and J. J. Moré (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming, Series B*, 91(2):201–213. doi: 10.1007/s101070100263.

E. D. Dolan, J. J. Moré, and T. S. Munson (2004). Benchmarking optimization software with COPS 3.0. Technical Report ANL/MCS-273, Argonne National Laboratory, Argonne, Illinois, USA.

I. S. Duff (2004). MA57—a code for the solution of sparse symmetric definite and indefinite systems. *ACM Transactions on Mathematical Software*, 30(2):118–144. doi: 10.1145/992200.992202.

I. S. Duff and J. K. Reid (1982). MA27—a set of Fortran subroutines for solving sparse symmetric sets of linear equations. Report AERE R10533, HMSO, London, UK.

R. Fletcher (1987). *Practical Methods of Optimization*. J. Wiley and Sons, Chichester, England, second edition. doi: 10.1002/9781118723203.

D. C.-L. Fong and M. A. Saunders (2011). LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971. doi: 10.1137/10079687X.

R. Fourer, D. M. Gay, and B. W. Kernighan (2002). *AMPL: A Modeling Language for Mathematical Programming*. Duxbury Press / Brooks/Cole Publishing Company, second edition.

R. Fourer and S. Mehrotra (1993). Solving symmetric indefinite systems in an interior-point method for linear programming. *Mathematical Programming*, 62(1-3):15–39. doi: 10.1007/bf01585158.

R. W. Freund (1993). A transpose-free quasi-minimal residual algorithm for non-hermitian linear systems. *SIAM Journal on Scientific Computing*, 14(2):470–482. doi: 10.1137/0914029.

M. P. Friedlander and D. Orban (2012). A primal–dual regularized interior-point method for convex quadratic programs. *Mathematical Programming Computation*, 4(1):71–107. doi: 10.1007/s12532-012-0035-2.

D. M. Gay (1997). Hooking your solver to AMPL. Technical Report 97-4-06, Lucent Technologies Bell Labs Innovations, Murray Hill, NJ. `www.ampl.com/REFS/HOOKING`.

D. M. Gay (2005). Writing `.nl` files. Technical Report SAND2005-7907P, Sandia National Laboratories, Albuquerque, NM.

E. M. Gertz and S. J. Wright (2003). Object-oriented software for quadratic programming. *ACM Transactions on Mathematical Software*, 29(1):58–81. doi: 10.1145/641876.641880.

R. Geus, D. Orban, and D. Wheeler (2009). PySparse: a fast sparse matrix library in Python. `pysparse.sf.net`.

Ph. E. Gill, W. Murray, and M. H. Wright (1981). *Practical Optimization*. Academic Press, London.

J. Gondzio and A. Grothey (2007). Parallel interior-point solver for structured quadratic programs: Application to financial planning problems. *Annals of Operations Research*, 152(1):319–339. doi: 10.1007/s10479-006-0139-z.

J. Gondzio and R. Sarkissian (2003). Parallel interior-point solver for structured linear programs. *Mathematical Programming*, 96(3):561–584. doi: 10.1007/s10107-003-0379-5.

N. I. M. Gould (1985). On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem. *Mathematical Programming*, 32(1):90–99. doi: 10.1007/BF01585660.

N. I. M. Gould, M. E. Hribar, and J. Nocedal (2001). On the solution of equality constrained quadratic programming problems arising in optimization. *SIAM Journal on Scientific Computing*, 23(4):1376–1395. doi: 10.1137/s1064827598345667.

N. I. M. Gould, D. Orban, and Ph. L. Toint (2003a). GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, 29(4):353–372. doi: 10.1145/962437.962438.

N. I. M. Gould, D. Orban, and Ph. L. Toint (2003b). CUTEr and SifDec, a Constrained and Unconstrained Testing Environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394. doi: 10.1145/962437.962439.

N. I. M. Gould, D. Orban, and Ph. L. Toint (2015a). An interior-point ℓ_1-penalty method for nonlinear optimization. In M. Al-Baali, L. Grandinetti, and A. Purnama, editors, *Recent Developments in Numerical Analysis and Optimization*, volume 134 of *Proceedings in Mathematics and Statistics*, pages 117–150, Switzerland. Springer. doi: 10.1007/978-3-319-17689-5. special issue of NAOIII, Muscat, Oman, 2014.

N. I. M. Gould, D. Orban, and Ph. L. Toint (2015b). CUTEst: a Constrained and Unconstrained Testing Environment with safe threads for Mathematical Optimization. *Computational Optimization and Applications*, 60:545–557. doi: 10.1007/s10589-014-9687-3.

N. I. M. Gould and Ph. L. Toint (2008). Nonlinear programming without a penalty function or a filter. *Mathematical Programming*, 122(1):155–196. doi: 10.1007/s10107-008-0244-7.

A. Griewank (2000). *Evaluating derivatives: Principles and techniques of algorithmic differentiation*. Number FR19 in Frontiers in Applied Mathematics. SIAM, Philadelphia, USA. doi: 10.1137/1.9780898717761.

W. E. Hart (2009). Coopr: A common optimization repository. Technical Report, Sandia National Laboratory, Albuquerque, NM. `software.sandia.gov/trac/coopr`.

W. E. Hart, C. Laird, J.-P. Watson, and D. L. Woodruff (2012). *Pyomo – Optimization Modeling in Python*. Springer US. doi: 10.1007/978-1-4614-3226-5.

Harwell Subroutine Library (2007). *A collection of Fortran codes for large-scale scientific computation*. AERE Harwell Laboratory, Harwell, Oxfordshire, England.

M. R. Hestenes and E. Stiefel (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436.

E. Jones, T. Oliphant, P. Peterson, et al. (2001). SciPy: Open source scientific tools for Python.

M. Kojima, N. Megiddo, and S. Mizuno (1993). A primal-dual infeasible-interior-point algorithm for linear programming. *Mathematical Programming*, 61(1-3):263–280. doi: 10.1007/bf01582151.

S. Leyffer (2004). MacMPEC: AMPL collection of MPECs. `www.mcs.anl.gov/~leyffer/MacMPEC`.

C.-J. Lin and J. J. Moré (1998). Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9:1100–1127. doi: 10.1137/S1052623498345075.

C.-J. Lin and J. J. Moré (1999). Incomplete Cholesky factorizations with limited memory. *SIAM Journal on Scientific Computing*, 21(1):24–45. doi: 10.1137/S1064827597327334.

D. Liu and J. Nocedal (1989). On the limited memory BFGS method for large scale optimization. *Mathematical Programming*, 45(1-3):503–528. doi: 10.1007/bf01589116.

A. Logg, K.-A. Mardal, G. N. Wells, et al. (2012). *Automated Solution of Differential Equations by the Finite Element Method*. Springer. doi: 10.1007/978-3-642-23099-8.

A. Logg and G. N. Wells (2010). DOLFIN: Automated finite element computing. *ACM Transactions on Mathematical Software*, 37(2):1–28. doi: 10.1145/1731022.1731030.

A. Makhorin (2006). *GNU Linear Programming Kit version 4.11*. Department for Applied Informatics, Moscow Aviation Institute, Moscow, Russia. `www.gnu.org/software/glpk/glpk.html`.

S. Mehrotra (1992). On the implementation of a primal-dual interior-point method. *SIAM Journal on Optimization*, 2(4):575–610. doi: 10.1137/0802028.

J. C. Meza, R. A. Oliva, P. D. Hough, and P. J. Williams (2007). OPT++: An object-oriented toolkit for nonlinear optimization. *ACM Transactions on Mathematical Software*, 33(2):12. doi: 10.1145/1236463.1236467.

J. J. Moré and D. J. Thuente (1994). Line search algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307. doi: 10.1145/192115.192132.

J. Nocedal (1980). Updating quasi-newton matrices with limited storage. *Mathematics of Computations*, 35:773–782.

J. Nocedal and Y. Yuan (1998). Combining trust region and line search techniques. In Y. Yuan, editor, *Advances in Nonlinear Programming*, pages 153–176, Dordrecht, The Netherlands. Kluwer Academic Publishers.

T. E. Oliphant (2006). *Guide to NumPy*. Provo, UT.

D. Orban (2009). PyKrylov: Krylov subspace methods in pure Python. `github.com/dpo/pykrylov`.

D. Orban (2014). Limited-memory ldl^t factorization of symmetric quasi-definite matrices with application to constrained optimization. *Numerical Algorithms*, 70(1):9–41. doi: 10.1007/s11075-014-9933-x.

C. C. Paige and M. A. Saunders (1975). Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629.

C. C. Paige and M. A. Saunders (1982a). Algorithm 583; LSQR: Sparse linear equations and least-squares problems. *ACM Transactions on Mathematical Software*, 8(2):195–209. doi: 10.1145/355993.356000.

C. C. Paige and M. A. Saunders (1982b). LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71. doi: 10.1145/355984.355989.

R. E. Perez, P. W. Jansen, and J. R. R. A. Martins (2012). pyOpt: A Python-based object-oriented framework for nonlinear constrained optimization. *Structures and Multidisciplinary Optimization*, 45(1):101–118. doi: 10.1007/s00158-011-0666-3.

A. Ronacher (2008). Jinja2: A full featured template engine for Python. `jinja.pocoo.org/`.

M. Sala, W. F. Spotz, and M. A. Heroux (2008). PyTrilinos: High-performance distributed-memory solvers for Python. *ACM Transactions on Mathematical Software*, 34(2):1–33. doi: 10.1145/1326548.1326549.

M. A. Saunders (1995). Solution of sparse rectangular systems using LSQR and CRAIG. *BIT Numerical Mathematics*, 35(4):588–604. doi: 10.1007/bf01739829.

R. B. Schnabel and E. Eskow (1999). A revised modified Cholesky factorization algorithm. *SIAM Journal on Optimization*, 9(4):1135–1148. doi: 10.1137/s105262349833266x.

P. Sonneveld (1989). CGS, a fast Lanczos-type solver for nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 10(1):36–52. doi: 10.1137/0910004.

T. Steihaug (1983). The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637. doi: 10.1137/0720042.

M. Stuart, M. OSullivan, and I. Dunning (2011). PuLP: a linear programming toolkit for Python. `www.optimization-online.org/DB_FILE/2011/09/3178.pdf`.

H. A. van der Vorst (1992). Bi-CGSTAB: A fast and smoothly converging variant of bi-CG for the solution of nonsymmetric linear systems. *SIAM Journal on Scientific and Statistical Computing*, 13(2):631–644. doi: 10.1137/0913035.

R. J. Vanderbei (1995). Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113. doi: 10.1137/0805005.

R. J. Vanderbei (2009). Nonlinear optimization models. `www.orfe.princeton.edu/~rvdb/ampl/nlmodels`.

S. Walter (2011a). ALGOPy. `github.com/b45ch1/algopy`.

S. Walter (2011b). PyADOLC. `github.com/b45ch1/pyadolc`.

S. Walter (2011c). PyCPPAD. `github.com/b45ch1/pycppad`.

A. Walther, A. Kowarz, and A. Griewank (2005). *Documentation of ADOL-C*. Updated version of Griewank et al., 1996.

CHAPITRE 8 DISCUSSION GÉNÉRALE

Dans cette thèse, deux méthodes d'optimisation non linéaire sans factorisation sont étudiées : une méthode de lagrangien augmenté et une méthode programmation quadratique séquentielle. Nous avons également présenté un environnement de développement d'algorithmes écrit en Python.

8.1 Synthèse des travaux

Les objectifs de cette thèse étaient de concevoir des algorithmes d'optimisation non linéaire ne nécessitant pas de factorisation afin de résoudre des problèmes de grande taille.

Dans le Chapitre 4 nous décrivons et implémentons un algorithme de lagrangien augmenté sans factorisation. Celui-ci ne nécessite que l'utilisation de produits de la jacobienne avec des vecteurs plutôt que de former la matrice au complet. De plus, il se compare favorablement à d'autres méthodes avec factorisation. L'utilisation d'approximations quasi-Newton pour le jacobien et le hessien dans une méthode lagrangien augmenté ainsi que l'application d'un algorithme sans factorisation sur un problème d'optimisation de structure d'une aile d'avion sont novateurs. Depuis la publication de notre article, notre algorithme a été utilisé avec succès sur des problèmes beaucoup plus réalistes. Par exemple, sur un problème de minimisation de la masse de la structure d'une aile d'avion dont la géométrie est celle d'un Boeing 777-200ER (Lambe et Martins, 2015).

Ensuite, une méthode de lagrangien augmenté proximal se fondant asymptotiquement en une méthode de programmation quadratique séquentielle stabilisée est développée dans le Chapitre 5. L'utilisation d'approximations BFGS à mémoire limitée du hessien du lagrangien confère aux systèmes linéaires rencontrés à chaque itération une structure de point de selle. Cette structure permet la reformulation des systèmes linéaires en des problèmes aux moindres carrés régularisés et préconditionnés. LSMR (Fong et Saunders, 2011) s'avère être le candidat naturel pour résoudre ces problèmes aux moindres carrés, notamment à cause de sa propriété de décroissance monotone du résidu de KKT. De plus, notre stratégie de résolution de ces systèmes permet l'usage de critères d'arrêts facilement implémentables et assurant une convergence globale et locale superlinéaire. Cette stratégie ne se limite toutefois pas au cadre des méthodes de programmation quadratique séquentielle et pourrait être utilisée au sein d'autres méthodes d'optimisation, par exemple dans les méthodes de points intérieurs.

Finalement, l'écosystème de développement d'algorithmes d'optimisation NLP.py est décrit

dans le Chapitre 7. Celui-ci fournit les blocs à partir desquels le chercheur peut élaborer des algorithmes plus sophistiqués, tels que ceux présentés dans les chapitres 4 et 5. Ces blocs comprennent des mécanismes de globalisation, des méthodes directes ou itératives pour l'algèbre linéaire, ainsi que quelques solveurs complets. NLP.py repose sur un module de modélisation très complet tenant compte de la grande diversité de provenance des dérivées et du choix de stockage des matrices. Les exemples d'application décrits dans le chapitre 7 démontrent l'utilité de NLP.py et la facilité avec laquelle ces algorithmes peuvent être implémentés. Nous sommes bien conscients que NLP.py n'est pas encore un produit fini. Cependant, les succès que nous rapportons dans le Chapitre 7, combinés avec les réactions positives reçues de la part des étudiants et des chercheurs depuis la présentation de NLP.py aux Journées de l'optimisation, montrent que c'est un environnement favorable au développement d'algorithmes d'optimisation.

8.2 Futures directions de recherche

Suivant les travaux développés dans cette thèse, quelques pistes d'améliorations possibles, de futures directions de recherche et extensions ont été mises en exergue.

A la lumière des bons résultats numériques présentés dans Armand et Omheni (2015), une amélioration des performances de notre algorithme du Chapitre 5 pourrait être obtenue en employant une stratégie différente de mise à jour du paramètre de pénalité dans les itérations internes. En effet, lorsque le paramètre de pénalité devient petit trop vite, il peut être nécessaire de faire un grand nombre d'itérations internes. Autoriser le paramètre de pénalité à augmenter permettrait d'éviter cet inconvénient tout en conservant la convergence globale de l'algorithme.

Un autre axe de recherche est l'extension de la méthode SQP régularisée proposée dans le chapitre 5 aux problèmes avec contraintes d'inégalité. Deux classes de méthodes semblent se présenter d'elles-mêmes : les méthodes SQP avec ensembles actifs et les méthodes de points intérieurs. Presque toutes les méthodes SQP pour les problèmes avec inégalités utilisent une méthode de programmation quadratique d'ensemble actif pour résoudre le sous-problème quadratique. Lorsqu'une approximation BFGS est employée, une itération mineure se résumerait à la résolution d'un système linéaire symétrique quasi-défini similaire à celui exposé dans le Chapitre 5 impliquant un sous-ensemble des variables et des contraintes. Une approche complètement différente pour traiter les contraintes d'inégalité est d'utiliser une méthode de points intérieurs dans laquelle chaque itération interne nécessiterait, elle aussi, la solution d'un système symétrique quasi-défini mais qui, cette fois, impliquerait toutes les contraintes du problème. Il ne parait cependant pas évident de déterminer laquelle de ces méthodes serait

la plus prometteuse lorsque combinée avec la résolution itérative inexacte de ces systèmes.

Il serait intéressant de comparer la performance des algorithmes qui découleraient de ces extensions avec celui présenté dans le Chapitre 4 sur des problèmes d'optimisation provenant de l'aérodynamique.

Enfin, NLP.py est en constante évolution. Rendre cet environnement plus visible permettrait de bénéficier de la rétroaction des usagers pour déterminer les nouvelles fonctionnalités à y ajouter et à en corriger les défauts. Un aspect qui, à mon sens, n'a pas été fortement développé dans la littérature concerne la parallélisation des algorithmes d'optimisation. NLP.py est un cadre propice pour ce genre de développement.

CHAPITRE 9 CONCLUSION

> «Pour tirer le meilleur parti des connaissances acquises, pour en extraire toute la richesse, il importe de ne pas s'y habituer trop vite, de se laisser le temps de la surprise et de l'étonnement.»
>
> Hubert Reeves

Ces propos d'Hubert Reeves illustrent bien le chemin emprunté lors cette thèse. Au cours des lectures, des recherches et de discussions avec Dominique et les autres collaborateurs des articles, notre compréhension de ce que doivent être les méthodes d'optimisation sans factorisation s'est transformée, clarifiée puis précisée pour finalement aboutir aux résultats qui ont été présentés dans les chapitres précédents.

Nous avons tout d'abord décrit une implémentation sans matrice et ainsi sans factorisation d'une méthode de lagrangien augmenté. Cet algorithme a été utilisé pour résoudre un problème de structure provenant de l'aérodynamique. Lors du développement de cet algorithme, nous nous sommes questionné sur l'adaptation sans factorisation d'autres méthodes d'optimisation possédant des taux de convergence plus rapides. Cette réflexion nous a mené à la conception d'un algorithme de lagrangien augmenté proximal se fondant en une méthode SQP asymptotiquement. Outre la convergence locale rapide, le point fort de cette approche consiste en l'utilisation d'approximations L-BFGS du hessien et en l'exploitation de l'étroite connexion entre les systèmes linéaires obtenus et les problèmes aux moindres carrés linéaires. Nous espérons que cet algorithme servira de base à de futurs travaux qui mèneront à des algorithmes sans factorisation plus généraux pouvant résoudre des problèmes de grande taille.

Nous avons finalement proposé un cadre de développement pour concevoir et implémenter plus facilement des méthodes d'optimisation. Nous désirons que cet environnement soit le plus utilisé possible tant par des chercheurs que par des étudiants désireux de découvrir l'optimisation.

RÉFÉRENCES

P. R. Amestoy, I. S. Duff et J.-Y. L'Excellent (1998). Multifrontal Parallel Distributed Symmetric and Unsymmetric Solvers. *Computer Methods in Applied Mechanics and Engineering*, 184:501–520. doi : 10.1007/bfb0095312.

R. Andreani, E. G. Birgin, J. M. Martínez et M. L. Schuverdt (2008). On augmented Lagrangian methods with general lower-level constraints. *SIAM Journal on Optimization*, 18(4):1286–1309. doi : 10.1137/060654797.

M. Arioli et D. Orban (2013). Iterative methods for symmetric quasi-definite linear systems. Part I: Theory. Rapport technique G-2013-32, GERAD, Montréal, Québec, Canada.

P. Armand et R. Omheni (2015). A globally and quadratically convergent primal-dual augmented Lagrangian algorithm for equality constrained optimization. *Optimization Methods and Software*. doi : 10.1080/10556788.2015.1025401. Online First.

S. Behnel, R. Bradshaw, C. Citro, L. Dalcin, D. Seljebotn et K. Smith (2011). Cython: The best of both worlds. *Computing in Science Engineering*, 13(2):31–39. doi : 10.1109/MCSE.2010.118.

D. P. Bertsekas (1973). Convergence rate of penalty and multiplier methods. Dans *1973 IEEE Conference on Decision and Control including the 12th Symposium on Adaptive Processes*. Institute of Electrical & Electronics Engineers (IEEE). doi : 10.1109/cdc.1973.269172.

D. P. Bertsekas (1976). Multiplier methods: A survey. *Automatica*, 12(2):133–145. doi : 10.1016/0005-1098(76)90077-7.

D. P. Bertsekas (1982). *Constrained Optimization and Lagrange Multiplier Methods*. Academic Press. doi : 10.1016/b978-0-12-093480-5.50005-2.

E. Birgin et J. Martínez (2014). *Practical Augmented Lagrangian Methods for Constrained Optimization*. SIAM, Philadelphia, USA. doi : 10.1137/1.9781611973365.

E. G. Birgin, D. Fernàndez et J. M. Martìnez (2012). The boundedness of penalty parameters in an augmented lagrangian method with constrained subproblems. *Optimization Methods and Software*, 27(6):1001–1024. doi : 10.1080/10556788.2011.556634.

P. T. Boggs et J. W. Tolle (1995). Sequential quadratic programming. *Acta Numerica*, 4:1–51. doi : 10.1017/s0962492900002518.

C. G. Broyden (1970). The convergence of a class of double-rank minimization algorithms 2. the new algorithm. *IMA Journal of Applied Mathematics*, 6(3):222–231. doi : 10.1093/imamat/6.3.222.

J. R. Bunch et L. C. Kaufman (1977). Some stable methods for calculating inertia and solving symmetric linear equations. *Mathematics of Computation*, 31:163–179. doi : 10.1090/s0025-5718-1977-0428694-0.

J. R. Bunch et B. N. Parlett (1971). Direct methods for solving symmetric indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 8(4):639–655. doi : 10.1137/0708060.

A. Buttari (2013). Fine-grained multithreading for the multifrontal qr factorization of sparse matrices. *SIAM Journal on Scientific Computing*, 35(4):323–345. doi : 10.1137/110846427.

J. D. Buys (1972). *Dual Algorithms for Constrained Optimization Problems*. Thèse de doctorat, University of Leiden, Leiden, the Netherlands.

R. H. Byrd, F. E. Curtis et J. Nocedal (2009). An inexact Newton method for non-convex equality constrained optimization. *Mathematical Programming*, 122(2):273–299. doi : 10.1007/s10107-008-0248-3.

R. H. Byrd, J. Nocedal et R. A. Waltz (2006). KNITRO: An integrated package for nonlinear optimization. Dans G. di Pillo et M. Roma, (édit.) : *Large-Scale Nonlinear Optimization*, pages 35–59. Springer Verlag.

A. R. Conn, N. I. M. Gould et P. L. Toint (1988). Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50(182):399–399. doi : 10.1090/s0025-5718-1988-0929544-3.

A. R. Conn, N. I. M. Gould et P. L. Toint (1992). *LANCELOT: A Fortran Package for Large-Scale Nonlinear Optimization (Release A)*. Springer-Verlag. doi : 10.1007/978-3-662-12211-2.

A. R. Conn, N. I. M. Gould et P. L. Toint (2000). *Trust-Region Methods*. SIAM, Philadelphia, USA. doi : 10.1137/1.9780898719857.

F. E. Curtis, Z. Han et D. P. Robinson (2014). A globally convergent primal-dual active-set framework for large-scale convex quadratic optimization. *Computational Optimization and Applications*, 60(2):311–341. doi : 10.1007/s10589-014-9681-9.

F. E. Curtis, J. Nocedal et A. Wächter (2010). A matrix-free algorithm for equality constrained optimization problems with rank-deficient jacobians. *SIAM Journal on Optimization*, 20(3):1224–1249. doi : 10.1137/08072471X.

W. C. Davidon (1991). Variable metric method for minimization. *SIAM Journal on Optimization*, 1(1):1–17. doi : 10.1137/0801001.

T. A. Davis (2004). Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method. *ACM Transactions on Mathematical Software*, 30(2):196–199. doi : 10.1145/992200.992206.

T. A. Davis (2011). Algorithm 915, SuiteSparseQR: Multifrontal Multithreaded Rank-revealing Sparse QR Factorization. *ACM Transactions on Mathematical Software*, 38(1):8:1–8:22. doi : 10.1145/2049662.2049670.

R. S. Dembo, S. C. Eisenstat et T. Steihaug (1982). Inexact newton methods. 19(2):400–408. doi : 10.1137/0719025.

J. E. Dennis et J. J. Moré (1974). A characterization of superlinear convergence and its application to quasi-Newton methods. *Mathematics of Computation*, 28(126):549–549. doi : 10.2307/2005926.

D. Fernández et M. Solodov (2010). Stabilized sequential quadratic programming for optimization and a stabilized newton-type method for variational problems. *Mathematical Programming*, 125(1):47–73. doi : 10.1007/s10107-008-0255-4.

R. Fletcher (1982). A model algorithm for composite nondifferentiable optimization problems. Dans *Mathematical Programming Studies*, pages 67–76. Springer Science + Business Media. doi : 10.1007/bfb0120959.

R. Fletcher et S. Leyffer (1998). User manual for filtersqp. *Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Dundee, Scotland*.

R. Fletcher et M. J. D. Powell (1963). A rapidly convergent descent method for minimization. *The Computer Journal*, 6(2):163–168. doi : 10.1093/comjnl/6.2.163.

D. C.-L. Fong et M. A. Saunders (2011). LSMR: An iterative algorithm for sparse least-squares problems. *SIAM Journal on Scientific Computing*, 33(5):2950–2971. doi : 10.1137/10079687X.

J. Gauvin (1977). A necessary and sufficient regularity condition to have bounded multipliers in nonconvex programming. *Mathematical Programming, Series B*, 12:136–138. doi : 10.1007/bf01593777.

P. E. Gill, W. Murray et M. A. Saunders (2005). SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Review*, 47(1):99–131. doi : 10.1137/S1052623499350013.

P. E. Gill et D. Robinson (2013). A globally convergent stabilized SQP method. *SIAM Journal on Optimization*, 23(4):1983–2010. doi : 10.1137/120882913.

P. E. Gill, M. A. Saunders et J. R. Shinnerl (1996). On the stability of Cholesky factorization for symmetric quasidefinite systems. *SIAM Journal on Optimization*, 17(1):35–46. doi : 10.1137/s0895479893252623.

P. E. Gill et E. Wong (2011). Sequential quadratic programming methods. Dans *Mixed Integer Nonlinear Programming*, pages 147–224. Springer Science + Business Media, New York, NY. doi : 10.1007/978-1-4614-1927-3_6.

D. Goldfarb, R. Polyak, K. Scheinberg et I. Yuzefovich (1999). A Modified Barrier-Augmented Lagrangian Method for Constrained Minimization. *Computational Optimization and Applications*, 14(1):55–74. doi : 10.1023/A:1008705028512.

N. I. M. Gould (1985). On practical conditions for the existence and uniqueness of solutions to the general equality quadratic programming problem. *Mathematical Programming*, 32(1):90–99. doi : 10.1007/bf01585660.

N. I. M. Gould et P. L. Toint (2000). SQP methods for large-scale nonlinear programming. Dans *System Modelling and Optimization*, pages 149–178. Springer Science + Business Media. doi : 10.1007/978-0-387-35514-6_7.

M. H. Gutknecht (2007). A brief introduction to Krylov space methods for solving linear systems. Dans *Frontiers of Computational Science*, pages 53–62. Springer. doi : 10.1007/978-3-540-46375-7_5.

W. W. Hager (1999). Stabilized sequential quadratic programming. Dans *Computational Optimization*, pages 253–273. Springer Science + Business Media. doi : 10.1007/978-1-4615-5197-3_13.

S. P. Han (1977). A globally convergent method for nonlinear programming. *Journal of Optimization Theory and Applications*, 22(3):297–309. doi : 10.1007/BF00932858.

Harwell Subroutine Library (2007). *A collection of Fortran codes for large-scale scientific computation*. AERE Harwell Laboratory, Harwell, Oxfordshire, England.

M. Heinkenschloss et D. Ridzal (2014). A matrix-free trust-region sqp method for equality constrained optimization. *SIAM Journal on Optimization*, 24(3):1507–1541.

M. R. Hestenes (1969). Multiplier and gradient methods. *Journal of Optimization Theory and Applications*, 4:303–320. doi : 10.1007/BF00927673.

M. R. Hestenes et E. Stiefel (1952). Methods of conjugate gradients for solving linear systems. *Journal of research of the National Bureau of Standards*, 49:409–436. doi : 10.6028/jres.049.044.

A. F. Izmailov et M. V. Solodov (2012). Stabilized SQP Revisited. *Mathematical Programming*, 133(1–2):93–120. doi : 10.1007/s10107-010-0413-3.

N. Karmarkar (1984). A new polynomial-time algorithm for linear programming. *Combinatorica*, 4(4):373–395. doi : 10.1007/BF02579150.

M. Kojima, S. Mizuno et A. Yoshise (1989). A primal-dual interior point algorithm for linear programming. Dans *Progress in Mathematical Programming*, pages 29–47. Springer Science + Business Media. doi : 10.1007/978-1-4613-9617-8_2.

M. Kočvara et M. Stingl (2011). PENNON: Software for linear and nonlinear matrix inequalities. Dans *Handbook on Semidefinite, Conic and Polynomial Optimization*, pages 755–791. Springer Science + Business Media. doi : 10.1007/978-1-4614-0769-0_26.

A. B. Lambe et J. R. R. A. Martins (2015). Matrix-free aerostructural optimization of aircraft wings. *Struct Multidisc Optim*, 53(3):589–603. doi : 10.1007/s00158-015-1349-2.

C. Lanczos (1952). Solution of systems of linear equations by minimized iterations1. *Journal of Research of the National Bureau of Standards*, 49(1).

S. Leyffer (2004). MacMPEC: AMPL collection of MPECs. `www.mcs.anl.gov/~leyffer/MacMPEC`.

J. J. Moré et D. Thuente (1994). Linesearch algorithms with guaranteed sufficient decrease. *ACM Transactions on Mathematical Software*, 20(3):286–307. doi : 10.1145/192115.192132.

B. A. Murtagh et M. A. Saunders (2003). MINOS 5.51 user's guide. Technical Report SOL 83-20R, Stanford University, Stanford, California, USA.

J. Nocedal, A. Sartenaer et C. Zhu (2002). On the behavior of the gradient norm in the steepest descent method. *Computational Optimization and Applications*, 22(1):5–35. doi : 10.1023/A:1014897230089.

J. Nocedal et S. J. Wright (2006). *Numerical Optimization*. Springer series in Operations Research. Springer-Verlag, 175 Fifth Avenue, New York, NY 10010, USA, 2ème édition.

D. Orban (2014). Limited-memory LDL^T factorization of symmetric quasi-definite matrices with application to constrained optimization. *Numerical Algorithms*, 70(1):9–41. doi : 10.1007/s11075-014-9933-x.

J. M. Ortega et W. C. Rheinboldt (2000). *Iterative Solution of Nonlinear Equations in Several Variables*. SIAM, Philadelphia, USA. doi : 10.1137/1.9780898719468.

C. C. Paige et M. A. Saunders (1975). Solution of sparse indefinite systems of linear equations. *SIAM Journal on Numerical Analysis*, 12(4):617–629. doi : 10.1137/0712047.

C. C. Paige et M. A. Saunders (1982a). Algorithm 583; LSQR: Sparse linear equations and least-squares problems. *ACM Transactions on Mathematical Software*, 8(2):195–209. doi : 10.1145/355993.356000.

C. C. Paige et M. A. Saunders (1982b). LSQR: An algorithm for sparse linear equations and sparse least squares. *ACM Transactions on Mathematical Software*, 8(1):43–71. doi : 10.1145/355984.355989.

M. J. D. Powell (1969). A method for nonlinear constraints in minimization problems. Dans R. Fletcher, éditeur : *Optimization*, pages 283–298. Academic Press, New York.

M. J. D. Powell (1978). A fast algorithm for nonlinearly constrained optimization calculations. Dans *Lecture Notes in Mathematics*, pages 144–157. Springer Science + Business Media. doi : 10.1007/bfb0067703.

R. T. Rockafellar (1973). The multiplier method of Hestenes and Powell applied to convex programming. *Journal of Optimization Theory and Applications*, 12:555–562. doi : 10.1007/BF00934777.

Y. Saad (2003). *Iterative Methods for Sparse Linear Systems*. SIAM, Philadelphia, USA, $2^{\text{ème}}$ édition.

M. A. Saunders (1996). Cholesky-based methods for sparse least squares: The benefits of regularization. Dans L. Adams et J. L. Nazareth, (édit.) : *Linear and Nonlinear Conjugate Gradient-Related Methods*, pages 92–100. SIAM, Philadelphia, USA.

K. Schittkowski (1982). The nonlinear programming method of Wilson, Han, and Powell with an augmented Lagrangian type line search function. *Numerische Mathematik*, 38(1):83–114. doi : 10.1007/BF01395810.

L. A. Schmit (1960). Structural Design by Systematic Synthesis. Dans *2^{nd} Conference on Electronic Computation*, pages 105–132, New York, NY. ASCE.

R. J. Vanderbei (1995). Symmetric quasi-definite matrices. *SIAM Journal on Optimization*, 5(1):100–113. doi : 10.1007/s11075-014-9933-x.

R. J. Vanderbei (1999). LOQO: An interior point code for quadratic programming. *Optimization Methods and Software*, 11(1-4):451–484. doi : 10.1080/10556789908805759.

A. Wächter et L. T. Biegler (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming, Series A*, 106(1):25–57. doi : 10.1007/s10107-004-0559-y.

Z. Wang, S.-C. Fang et W. Xing (2013). On constraint qualifications: Motivation, design and inter-relations. *Journal of Industrial and Management Optimization*, 9(4):983–1001. doi : 10.3934/jimo.2013.9.983.

R. B. Wilson (1963). *A Simplicial Method for Convex Programming*. Thèse de doctorat, Harvard University, Boston, USA.

M. H. Wright (1992). Interior methods for constrained optimization. *Acta numerica*, 1:341–407. doi : 10.1017/s0962492900002300.

S. J. Wright (1998). Superlinear convergence of a stabilized SQP method to a degenerate solution. *Computational Optimization and Applications*, 11(3):253–275. doi : 10.1023/A:1018665102534.